Facultad de Ingeniería

Ingeniería de Sistemas e Informática

"IMPLEMENTACIÓN DE HERRAMIENTAS DE INTEGRACIÓN CONTINUA PARA GESTIÓN DE CÓDIGO FUENTE EN PANDORA TECHNOLOGIES"

Bachiller:

Monica Lizeth Guido Saravia

para optar el Título Profesional de Ingeniero de Sistema e Informática

Lima – Perú

2018

Dedicatoria:

El presente proyecto está dedicado en primer lugar a mis padres Julio y Gladys quienes con mucho sacrificio me dieron la oportunidad de terminar mi carrera profesional, gracias por inculcarme valores y siempre apoyarme moralmente. Dedicado también a mis hermanos quienes siempre me apoyaron incondicionalmente a lo largo de mi etapa universitaria.

Agradecimiento:

Agradecer infinitamente a toda mi familia y amigos quienes de alguna manera me apoyaron para culminar el presente proyecto. A mi asesor quien desde el inicio del PET se hizo parte de nuestros proyectos ayudando a culminarlo con éxito.

Resumen

El presente proyecto se realizó con el principal objetivo de optimizar la gestión del proceso de actualización de código fuente así como estandarizar y asegurar la calidad del mismo.

Para lograr el objetivo ha sido necesaria la utilización de Herramientas de Integración Continua, los cuales permiten poder llevar un mejor control de las fuentes y por conseguiente detectar errores de manera oportuna reduciendo costos a la empresa al tener obligatoriamente que corregir los errores para evitar que las aplicaciones desarrolladas sean rechazadas por el usuario final.
Se ha analizado detalladamente cada una de las etapas del ciclo de desarrollo de software para poder seleccionar las herramientas correctas que nos van a permitir cumplir con los objetivos trazados.

Existen, hoy en dia, numerosas soluciones que se adaptan a distintas necesidades y que presentan diferentes costes y requerimientos. Es por este motivo que es importante analizar bien la oferta y determinar cual es la herramienta más adecuada a cada caso.

Contenido

ÍNDICE DE FIGURAS

Introducción

El alcance del presente proyecto es el diseño del nuevo procedimiento de actualización de código y la implantación de herramientas de integración continua: GIT, Bitbucket, SonarQube y Jenkins para estandarizar la calidad del desarrollo de software y la gestión del proceso de actualización de código fuente en las áreas de desarrollo, aseguramiento de la calidad (QA) y Producción.

El desarrollo del proyecto también ayuda a mitigar los riesgos de desarrollo de software como son: baja calidad del software, detección tardía de fallas en la aplicación y la pérdida de visibilidad del proyecto. De la misma manera se logró reducir los procesos repetitivos como la compilación de código fuente, pruebas, inspección y retroalimentación reduciendo así costos, tiempo y esfuerzo.

Actualmente las organizaciones necesitan del desarrollo de aplicaciones o software que les ayuden a lograr sus objetivos, debido a que las demandas del mercado requieren de software capaces de afrontar los cambios. Es por este motivo que es indispensable contar con un software que nos ayude a responder al dinamismo de la empresa.

CAPÍTULO 1

ASPECTOS GENERALES

1.1 Definición del Problema

1.1.1 Descripción del Problema

Pandora Technologies es una empresa peruana desarrolladora de Soluciones de Software y de Aplicaciones Web a la medida, además de brindar un servicio especializado de testing, para empresas y todo tipo de organizaciones. Su propuesta de valor se basa en la entrega de soluciones de software personalizadas, capaces de soportar procesos clave y actividades de misión crítica, alineadas con los objetivos estratégicos de los clientes.

Pandora se compromete a proporcionar soluciones de productividad de pruebas que ayudan a las empresas a mantener y mejorar sus aplicaciones de software con mayor eficacia.

El presente proyecto de “Implementación de Sistema de Integración Continua para Gestión de código fuente en Pandora” surge para agilizar procesos de integración de diversos proyectos y aplicaciones. Además para que los desarrolladores trabajando en un proyecto puedan integrar los cambios en sus códigos al menos una vez al día y que cada integración sea verificada para detectar errores tan rápido como es posible.

El desarrollo del proyecto ayudó a mitigar los riesgos de un proyecto de software como son: pérdida de cohesión, detección tardía de los defectos, baja calidad del software y la pérdida de visibilidad del proyecto. De la misma manera se logró reducir los procesos repetitivos como la compilación de código fuente, pruebas, inspección y retroalimentación reduciendo así costos, tiempo y esfuerzo.

Se estableció una mayor confianza del producto, la aplicación práctica y efectiva de la integración continua proporciona una mayor confianza de un producto de software, debido a que por cada construcción, el equipo conoce las pruebas que son ejecutadas contra el software y permite verificar su comportamiento, conocer los estándares de diseño y código que reúnen y por ende conocer el resultado funcional del producto.

Árbol del Problema:

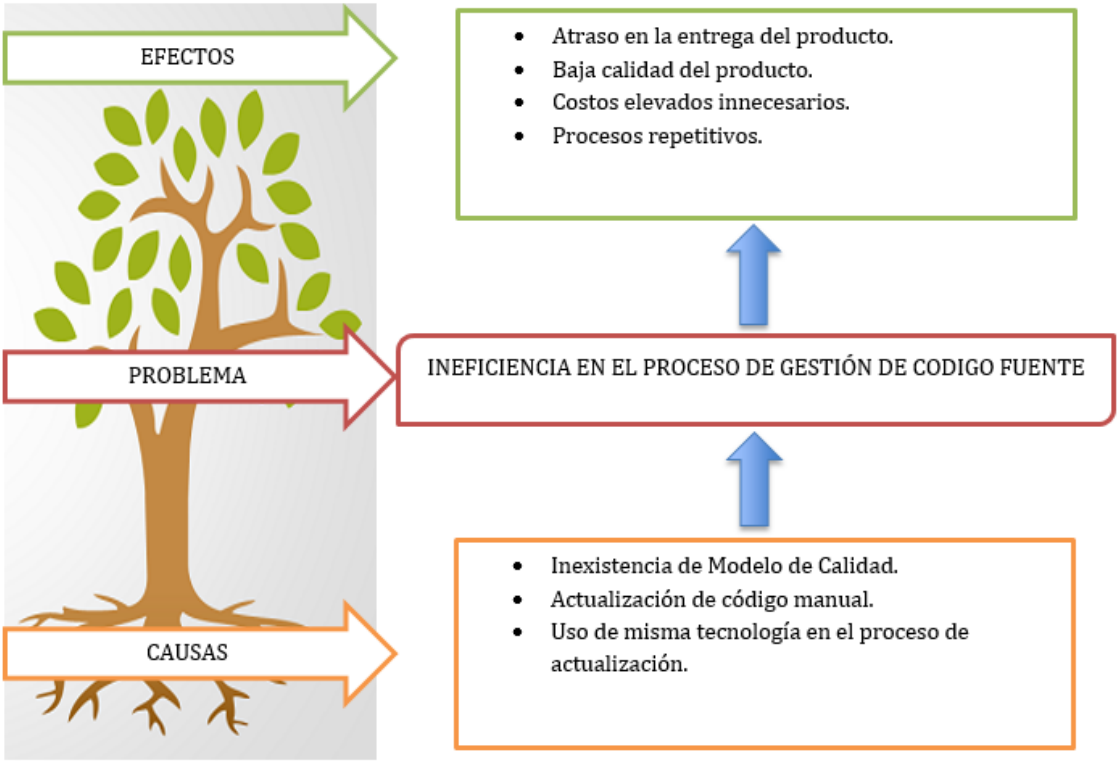


Figura Nº1 (Diagrama del árbol)

Figura Nº2 (Tabla del problema)

1.1.2 Formulación del Problema

1.1.2.1 Problema Principal:

Luego del análisis realizado a la situación actual del proceso de actualización de código en las diferentes áreas de la organización, se ha determinado que el problema es:

¿Cómo se puede aumentar la eficiencia en el proceso de gestión de código fuente mediante la implantación de Herramientas de Integración Continua?

1.1.2.2 Problemas Específicos

- Inexistencia de modelos de calidad en los sistemas de la empresa, procedimiento de control de fuentes no definido.
- El analista realiza la integración de código de manera manual abarcando tiempo muy valioso que puede ser usado en las pruebas de calidad.
- La empresa utiliza herramientas deficientes en el desarrollo de software que no permiten detectar los errores en etapa temprana.

1.2 Definición de Objetivos

1.2.1 Objetivo General

Mejorar la eficiencia en el proceso de actualización y versionamiento de código fuente para el desarrollo de software mediante la integración continua para su posterior automatización.

1.2.2 Objetivos Específicos

- ¿De que manera la organización realiza el proceso de actualización de código fuente en las áreas de: Desarrollo, QA y Producción para estandarizar la calidad de código fuente en el proceso operativo para reducir tiempo y costo?
- ¿Qué plantillas de reglas de desarrollo o modelos de calidad se pueden elaborar para permitir estandarizar el código fuente, detectar y reducir los errores de desarrollo de software?
- ¿Cómo mejorar la eficiencia del proceso de integración de código fuente con el apoyo de las herramientas de IC: Bitbucket, GIT, Sonar y Jenkins e incrementar de esta manera el nivel de calidad de código?

1.2.3 Alcances y Limitaciones

1.2.3.1 Alcances

El alcance del presente proyecto es la implantación de herramientas de integración continua: GIT, Bitbucket, SonarQube y Jenkins para estandarizar la calidad del desarrollo de software y la gestión del proceso de actualización de código fuente en las áreas de desarrollo, aseguramiento de la calidad (QA) y Producción.

La implantación de las herramientas de integración continua contempla las fases de análisis y diseño del proceso operativo de actualización y versionamiento de código fuente así como la instalación y configuración de las mismas que permiten obtener la versión funcional operativa del sistema de integración continua propuesto.

Cada herramienta instalada contara como máximo con 10 usuarios quienes accederán a las aplicaciones para la administración de los requerimientos en el proceso de desarrollo del aplicativo.

El sistema de integración continua será entregado con los Manuales de usuario respectivos, se contempla 1 mes de capacitaciones a los colaboradores que requieran realizar actualización de código fuente de producción, así como a las distintas partes interesadas que intervienen en la atención, directa o indirecta, del desarrollo de software.

1.2.3.2 Limitaciones

El diseño del proceso de actualización de código fuente no tiene como alcance la recuperación de información en el caso de pérdida debido al mal uso de las herramientas por parte del usuario.

La implantación de las herramientas SonarQube y Jenkins no contempla la corrección automática de errores que se puedan identificar en el código generado por la fábrica de software o el área de desarrollo de la organización.

1.2.4 Justificación

El presente proyecto es de mucha importancia dado que permite constatar como las técnicas de la integración continua orientan a mejorar el desarrollo de software, conduciendo a todo el equipo de desarrollo proponerse a entregar un producto de software de buena calidad y que cumpla con los requisitos de los usuarios y del negocio.

La investigación desarrollada busca la interiorización de la integración continua en todo el equipo de desarrollo y que tomen conciencia de la importancia de lograr un buen control de desarrollo de software a través de la integración del código fuente, las pruebas continuas, la inspección del código fuente y la retroalimentación continua.

La integración continua en la actualidad aparece como una opción de solución a los diferentes problemas que cada día afronta el desarrollo de software, ayudando a mitigar los problemas de software identificando constantemente y de manera temprana los errores en base a las pruebas y la inspección del software.

1.2.5 Estado del Arte

1.2.5.1 Herramientas de integración continua

Entre las herramientas con las que se dispone hoy en día para integración continua se muestran las siguientes:

- **GIT**

  Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Al principio Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

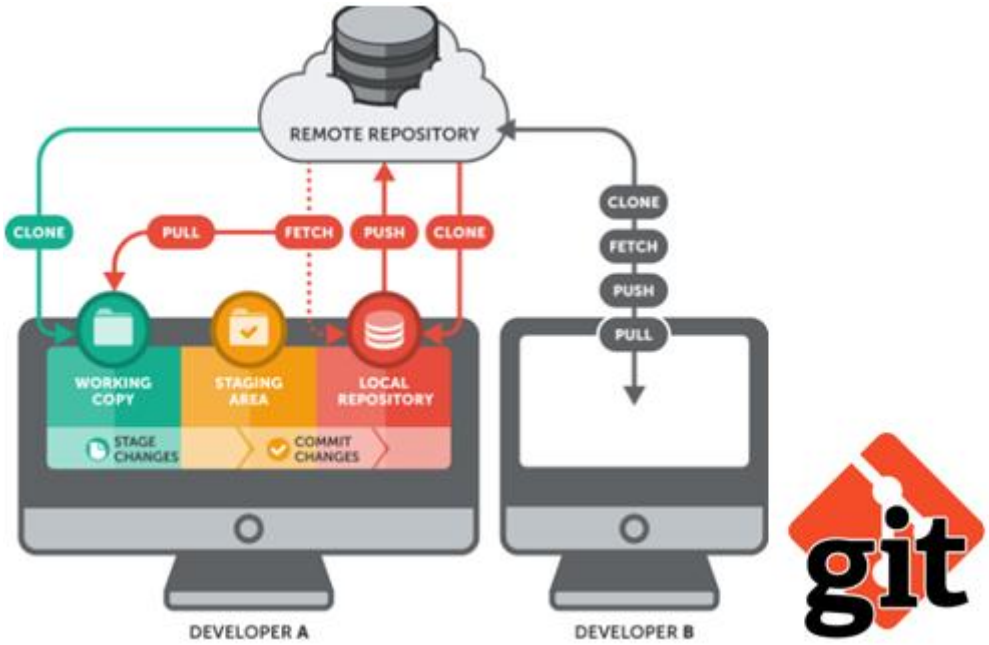


Figura Nº3 (Herramienta de IC - GIT)

- **JENKINS**

  Jenkins proporciona integración continua para el desarrollo de software. Es un sistema corriendo en un servidor que es un contenedor de servlets, como Apache Tomcat. Soporta herramientas de control de versiones como CVS, Subversion, Git, Mercurial, Perforce y Clearcase y puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y programas batch de Windows.

  Esta herramienta, proviene de otra similar llamada Hudson, ideada por Kohsuke Kawaguchi, que trabajaba en Sun. Unos años después de que Oracle comprara Sun, la comunidad de Hudson decidió renombrar el proyecto a Jenkins, migrar el código a Github y continuar el trabajo desde ahí. No obstante, Oracle ha seguido desde entonces manteniendo y trabajando en Hudson.

Figura Nº4 (Herramienta de IC - Jenkins)

- **SONARQUBE**

  SonarQube (conocido anteriormente como Sonar) es una plataforma para evaluar código fuente. Es software libre y usa diversas herramientas de análisis estático de código fuente como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.

  Entre sus principales funciones:

  Informa sobre código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, complejidad ciclomática, potenciales errores, comentarios y diseño del software.

  Se integra con Maven, Ant y herramientas de integración continua como Atlassian Bamboo, Jenkins y Hudson.

Figura Nº5 (Herramienta de IC - SonarQube)

- **BITBUCKET**

  Bitbucket es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de revisiones Mercurial y Git. Bitbucket ofrece planes comerciales y gratuitos. Se ofrece cuentas gratuitas con un número ilimitado de repositorios privados (que puede tener hasta cinco usuarios en el caso de cuentas gratuitas) desde septiembre de 2010, los repositorios privados no se muestran en las páginas de perfil - si un usuario sólo tiene depósitos privados, el sitio web dará el mensaje "Este usuario no tiene repositorios". El servicio está escrito en Python.

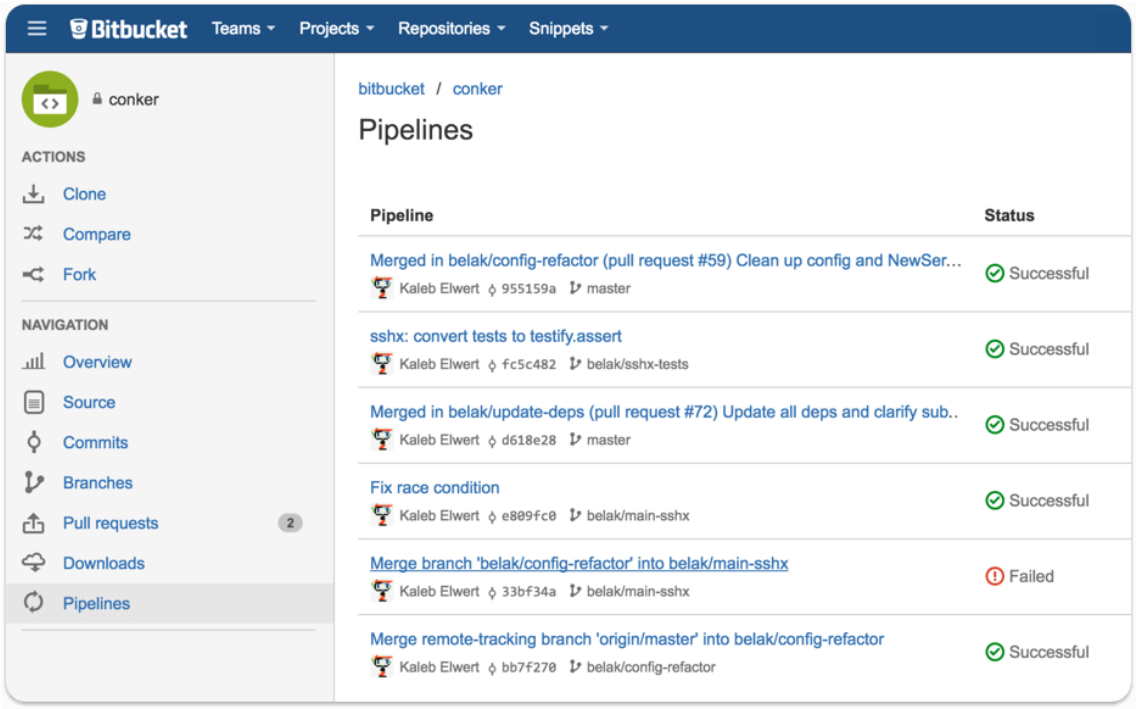


Figura Nº6 (Herramienta de IC - Bitbucket)

1.2.5.2 Ventajas

Las ventajas de usar herramientas de integración continua en el proceso de desarrollo de software son:

- Posibilidad de monitorizar el control de versiones y actuar rápidamente ante cualquier cambio para evitar conflictos entre las versiones utilizadas para generar los entregables.
- Permite detectar en pocos minutos errores introducidos por cambios en el código, gracias al que el código se compila cada vez que se hace el commit en el servidor de integración continua.
- Al ejecutar las pruebas unitarias cada vez que se ejecuta el build, permite al desarrollador darse cuenta de cualquier error en su código.
- Permite visualizar el histórico de los resultados de las compilaciones, esto nos ayuda a ver cómo ha evolucionado la cobertura de código de los tests, complejidad y líneas de código.
- Ante cualquier error de conflicto por el pase de varios requerimientos, las herramientas permiten poder revertir los cambios con mucha facilidad y rapidez, a diferencia de SVN que se debe revertir objeto por objeto perdiendo mucho tiempo.

CAPÍTULO 2

MARCO TEÓRICO

2.1 Fundamento Teórico

2.1.1 Integración continua

2.1.1.1 IC según Martin Paoletta CIO de Redbee

Redbee es una firma de consultoría y servicios muy conocida encargada de ayudar a las empresas a acelerar su proceso de transformación digital, en esta empresa se tomó la decisión de utilizar la integración continua en el cual la definen como una práctica para realizar despliegues incrementales que se basan en funcionalidad lo que los desarrolladores denominan “liberar código” los cuales son validados por ambientes de preproducción o área de aseguramiento de la calidad.

“Si se asegura cierto nivel de calidad, esa funcionalidad puede ponerse en producción inmediatamente. Este proceso está automatizado casi en su totalidad, minimizando de esta manera errores humanos”, explica Martin Paoletta, CIO de Redbee.

La integración continua en esta empresa rompió el paradigma del ciclo de desarrollo – prueba – despliegue añadiendo modificaciones de todo el equipo para que el software vaya creciendo de manera incremental y que siempre esté listo para ser puesto en producción a dar valor, para un desarrollador de Redbee terminar con sus tareas ya no se trata de “terminé el desarrollo de la funcionalidad” sino que ya está en producción y dando valor al negocio.

Paoletta explica que la integración continua reduce considerablemente el tiempo de llegada al mercado comparando un ciclo de desarrollo ágil normal basado en iteraciones que tardan aproximadamente un mes donde se incorporan el trabajo de todo un equipo de desarrollo formando un conjunto grande de cambios que luego debe ser validado y certificado por un área de calidad y en muchas ocasiones añadiendo la iteración de pruebas y realización de correcciones contra el desarrollo de una funcionalidad que tarda aproximadamente dos días, se valida automáticamente y se despliega. En Redbee la decisión de utilizar integración continua redujo un proceso manual de cinco días a veinte minutos.

2.1.2 Prácticas de la integración continua

A continuación se detallan las principales prácticas de la IC:

1. **Mantener siempre un repositorio de código fuente.**

El desarrollo de software implican a un conjunto de ficheros que necesitan ser organizados para elaborar un producto, así mismo hacer el debido seguimiento de ellos implica altos costos y más aún si se involucran varias personas para realizar la actividad. Como solución a este problema existe la herramienta de gestión de código “**GIT”** que además forma parte de las herramientas implementadas en el presente proyecto.

2. **Automatizar la construcción.**

La construcción de un software implica compilar el código, para ganar tiempo y evitar muchos errores, esta tarea puede ser automatizada por dos herramientas como son Maven o Ant.

3. **Entregar todos los días el código actualizado a la línea principal.**

La integración les permite a los desarrolladores estén informados de los cambios que se realizan en el día, es importante realizar esto de manera frecuente para que todo el equipo sepa rápidamente que cambios se han realizado.

4. **Crear la línea base o principal desde la máquina de integración.**

Pese a que los desarrolladores realicen la entrega diaria de código, pueden ocurrir varios errores de integración debido a muchas causas entre ellas la falta de disciplina del equipo.

Para evitar este tipo de errores se debe estar seguro que se está construyendo el proyecto en la máquina de integración, la construcción se puede llevar a cabo de manera manual o a través de un servidor de IC como es el caso de Jenkins.

5. **Realizar las pruebas en un ambiente que sea réplica del entorno de producción.**

El riesgo de que ocurran errores en desarrollo y no en producción o viceversa es mayor si se realizan las pruebas en un ambiente con una versión distinta a la de producción. Para reducir estos riesgos es recomendable homologar en ambos entornos la misma versión de archivos de configuración, laas librerías a usar, etc.

6. **Lograr que todo el equipo de sistemas tenga disponible la última versión de código de manera fácil y rápida.**

Toda persona que esté involucrada en el desarrollo de un software en específico debe poder obtener fácilmente a la última versión de código y poder ejecutarlo con el fin de facilitar la revisión de los últimos cambios.

7. **Dar a conocer a todos que está pasando en el proceso de desarrollo del software.**

La comunicación en la integración continua es un factor muy importante, por lo que se debe asegurar que todo el equipo involucrado en el proyecto de software pueda visualizar fácilmente cuál es el estado en el que se encuentra el sistema y los distintos cambios que se van produciendo.

Algo muy importante que se debe comunicar es el estado en el que se encuentra la línea base o principal de desarrollo. Para esto existe la herramienta SonarQube que trabaja en conjunto con el servidor de integración continua y permite informar al equipo el progreso de los trabajos en cada momento y de manera muy clara.

8. **Automatizar el despliegue.**

La integración continua necesita de varios entornos para llevarse a cabo, como todos conocemos las áreas comunes son Desarrollo, aseguramiento de la calidad y producción quienes son las que interactúan diariamente y por lo tanto el proceso operativo debe ser más engorrosos para esto es mejor que los despliegues sean automatizados mediante scripts y hacer esta actividad de manera sencilla.

2.1.3 Beneficios de la integración continua

La reducción del riesgo es uno de los principales beneficios de utilizar la integración continua, dado a que es posible al analista de alguna manera conocer el tiempo a emplear para la integración debido a que las actividades son realizadas continuamente.

Entre otros beneficios que aporta la integración continúa encontramos la reducción de aparición de bugs o errores, puesto que el seguimiento y la realización de pruebas constante permiten detectarlos y corregirlos rápidamente antes de que entren a producción.

2.1.4 Metodología

Para el presente trabajo la metodología empleada es propia de la empresa Pandora Technologies, metodología a la cual se le denomina “Metodología PDRT”, a continuación se grafica las fases para el proceso de desarrollo de la solución implementada:

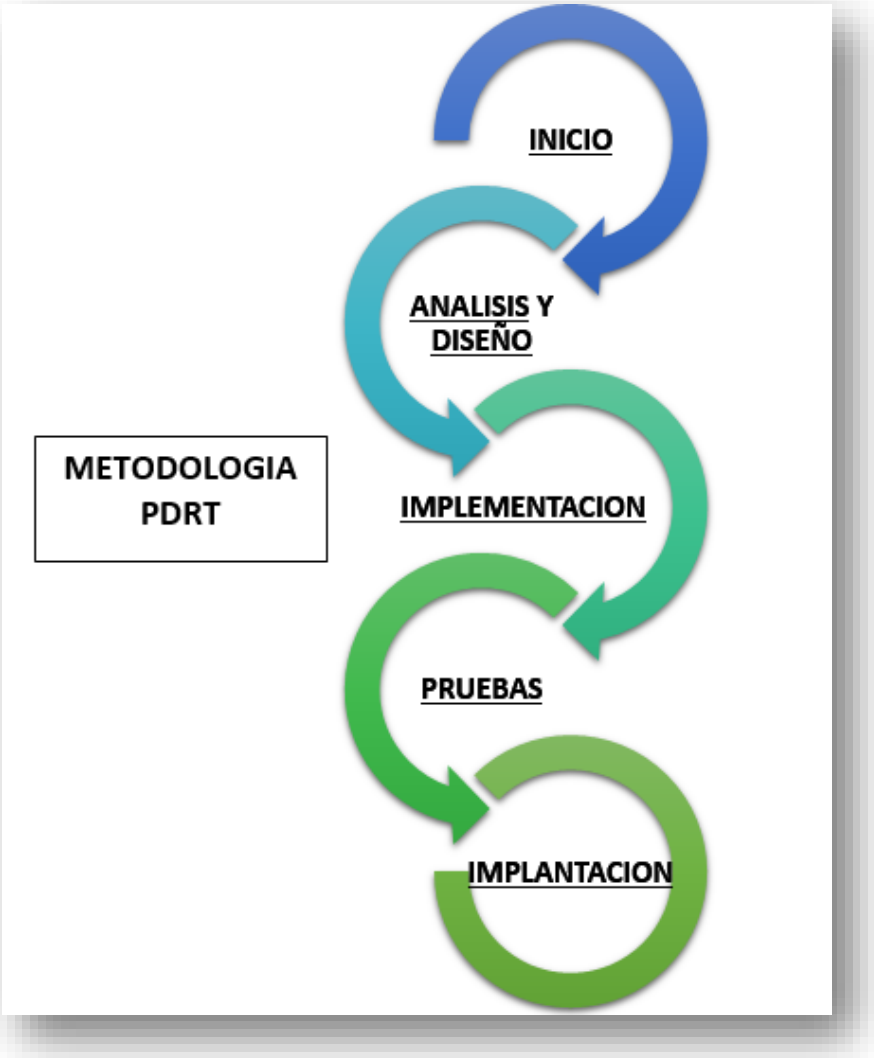


Figura Nº7 (Metodología PDRT)

Cada una de las fases de la metodología PDRT poseen los siguientes criterios:

1. **Inicio:**
   - ✓ Levantamiento de información
   - ✓ Requisitos
2. **Análisis y Diseño:**
   - ✓ Análisis en las distintas necesidades y requerimientos del sistema
   - ✓ Diseño del nuevo sistema desarrollado.
3. **Implementación:**
   - ✓ Instalación de las herramientas.
   - ✓ Configuración de las herramientas implementadas.
   - ✓ Elaboración de las plantillas de estándar de calidad de código.
   - ✓ Elaboración de los Manuales de procedimiento.
4. **Pruebas:**
   - ✓ Pruebas Unitarias
   - ✓ Pruebas QA
   - ✓ Pruebas de usuario
5. **Implantación:**
   - ✓ Puesta en producción del nuevo sistema
   - ✓ Evaluación del sistema
   - ✓ Soporte a la producción

El detalle de las actividades realizadas en cada fase de la metodología mencionada previamente son:

Inicio.- En esta primera fase el analista se involucra en la identificación de los problemas, oportunidades y objetivos. Esta fase es muy importante para el éxito del proyecto, puesto que la solución brindada estará basado en el problema identificado y de no detectar el problema real, la solución sería un fracaso.

En esta etapa también el analista determina los requerimientos de información en conjunto con los usuarios involucrados.

Análisis y Diseño.- Analizar las necesidades o requerimientos del sistema, con la ayuda de herramientas y/o técnicas especiales, pueden incluir el uso de diagramas de flujos de datos, etc.

El analista utiliza la información recolectada en la fase previa y elabora el diseño del nuevo flujo del sistema, este nuevo flujo es elaborado en conjunto con el usuario quien aprueba finalmente el diseño realizado.

Implementación.- El analista trabaja en la instalación y configuración de herramientas necesarias para desarrollar el nuevo sistema en base al diseño realizado previamente y aprobado por el usuario.

En esta etapa es donde el analista ejecuta los requerimientos y ayuda a los usuarios a elaborar los documentos indispensables del sistema.

Pruebas.- El sistema debe probarse antes de ser utilizado, esto ayuda a detectar los problemas o errores antes de la entrega del sistema o solución y reducir costos.

El analista ejecuta pruebas unitarias, así mismo se realizan pruebas de calidad ejecutadas por el área de aseguramiento de la calidad (QA), quienes realizan pruebas eshaustivas con el fin de encontrar errores o fallas en el sistema y finalmente se llevan a cabo las pruebas de usuarios quienes determinan si el nuevo sistema o solución está apto para ser puesto en producción.

Implantación.- Esta etapa es muy importante y bastante critica puesto que el analista se encarga de enviar el paquete a producción para su implantación, así como realizar la evaluación del sistema luego de la puesta en producción.

2.1.5 Características de la IC

Las principales características de la IC se describen a continuación:

- **Compilación de código fuente**

  En la integración continua esta característica es una de las más comunes pero muy importante de un sistema de IC.

- **Pruebas**

  Las pruebas son fundamentales en la realización del desarrollo de software puesto que asegura la calidad del producto.

- **Inspección**

  Se basa en las mejores prácticas o normas del desarrollo con el objetivo de validar la calidad del software y de programación, para esto se elaboran plantillas de calidad basada en estándares propuestos por el área de sistema.

- **Despliegue**

El sistema de IC tiene como objetivo la generación de los artefactos de software actualizados con los últimos cambios y su disposición en los ambientes o entornos correspondientes ya sea desarrollo, Calidad y Producción.

- **Documentación**

  Esta característica es de mucha importancia dado que la integración continua se basa en el código fuente del repositorio de control de versiones para actualizar constantemente la documentación del software.

- **Retroalimentación continua**

  La retroalimentación proporciona información del estado en el que se encuentra el software para la toma de decisiones.

2.1.6 Glosario de términos

En seguida se describe el significado de cada uno de los términos empleados en el Marco teórico:

- **Despliegue**

  Poner en práctica una actividad, concretar una exhibición o demostración.

- **CIO**

  Oficial en jefatura del área de sistemas de una organización.

- **Maven**

  Es una herramienta de software para la gestión y construcción de proyectos Java.

- **Ant**

  Es una herramienta usada en programación para la realización de tareas mecánicas repetitivas, normalmente durante la fase de compilación y construcción.

- **Bug**

  Error encontrado en un programa informático.

- **Repositorio de control de versiones**

  Es un sistema que permite almacenar diferentes versiones de un archivo, generalmente código fuente y librerías.

- **Ficheros**

  Es un conjunto de bits que son almacenados en un dispositivo.

- **Funcionalidad**

  Son las funciones de un sistema de información.

- **Automatizar**

  Aplicar procedimientos automáticos en la realización de un proceso o en una industria.

- **Pruebas**

  Es el proceso de verificar y revelar la calidad de un software.

- **Inspección**

  Es el análisis de código fuente con respecto a la calidad interna del software.

- **Proceso**

  Secuencia de pasos ejecutados con un propósito específico.

CAPÍTULO 3

DESARROLLO DE LA SOLUCIÓN

Para lograr el principal objetivo de Mejorar la eficiencia en el proceso de actualización y versionamiento de código fuente en la organización se ha procedido a ejecutar cada uno de los objetivos específicos los cuales se detallan a continuación.

Objetivo Especifico Nº1:

Analizar la manera de cómo la organización realiza el proceso de actualización de código fuente en las áreas de: Desarrollo, QA y producción, automatizar el proceso para reducir tiempo y costo.

En la empresa se maneja el siguiente proceso manual de actualización de código fuente:

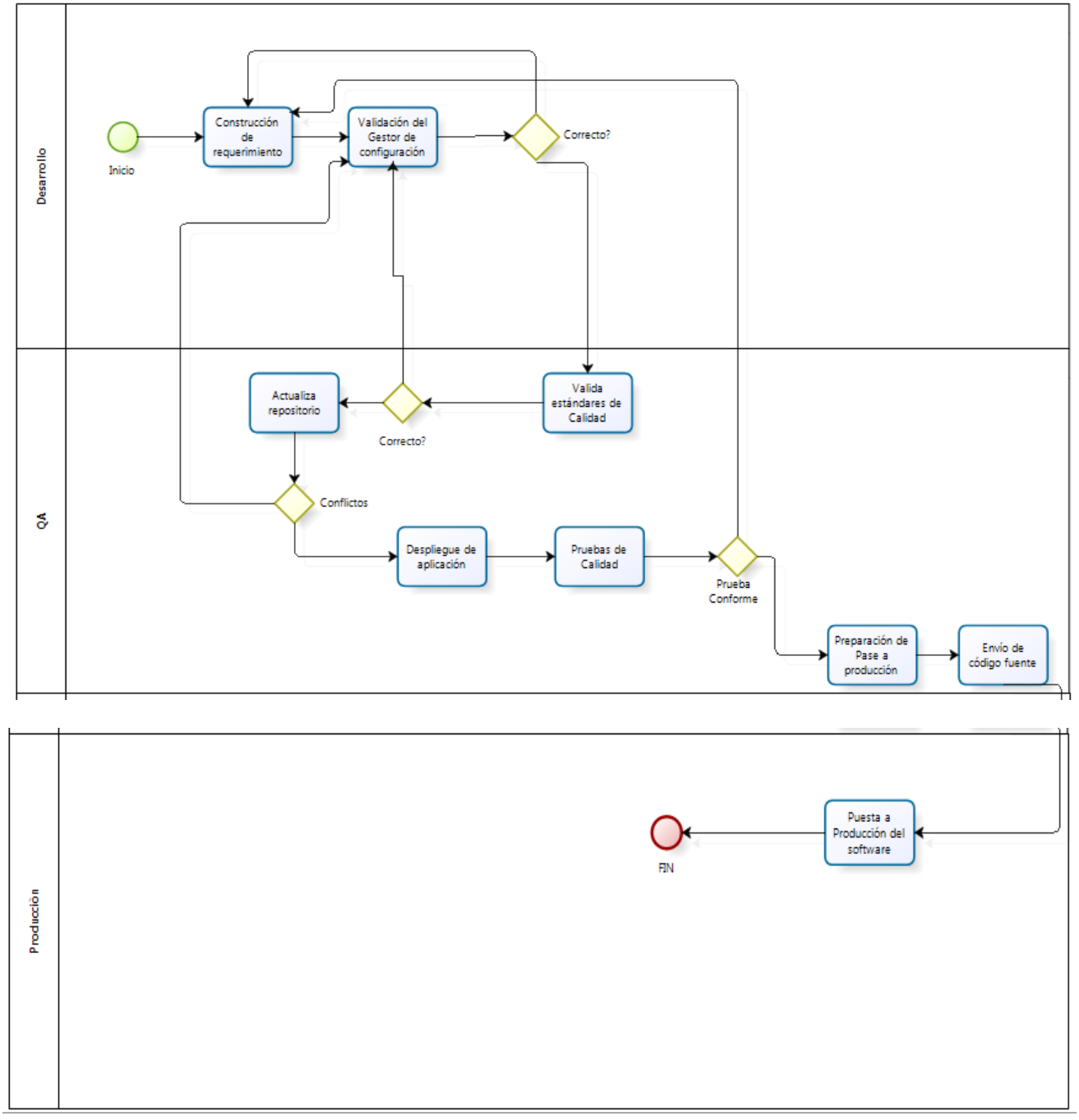


Figura Nº8 (Proceso de Actualización de código fuente)

El tiempo aproximado utilizado para la actualización de código para un proyecto “X” desde el área de fábrica o desarrollo hasta el pase a QA era de 4 días, con el nuevo procedimiento elaborado de acuerdo a las actividades de cada área se ha logrado reducir el tiempo del proceso actualización a 2 horas.

Se han planteado procesos de actualización de código fuente en cada una de las áreas antes mencionadas, los procesos se han realizado en base a la utilización de las herramientas de integración continua: GIT, Bitbucket, SonarQube y Jenkins.

Para el control de código fuente se tienen varios escenarios.

Escenario I.- Control de fuentes para requerimientos, primer envío hasta su pase a producción

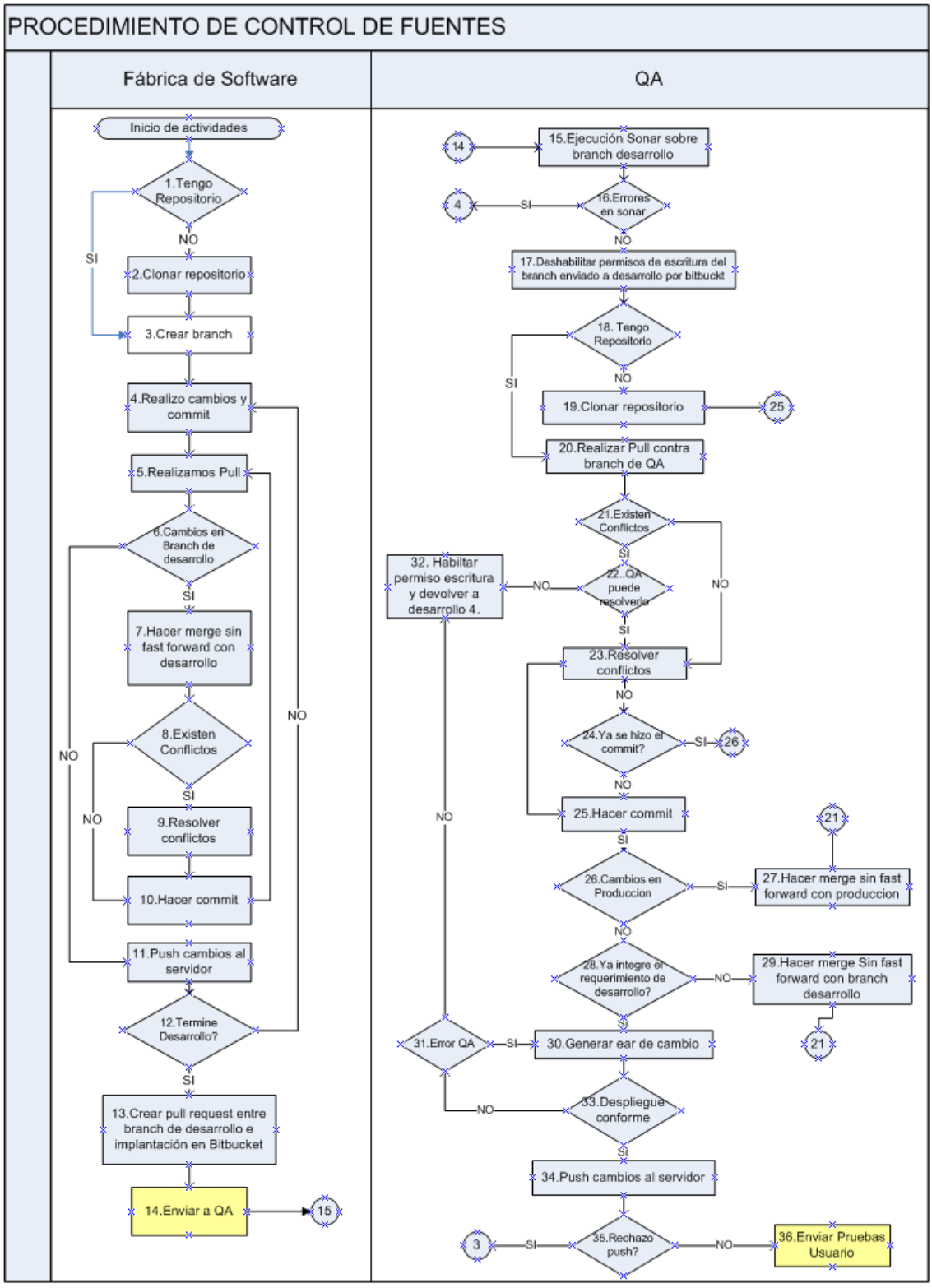


Figura Nº9 (Escenario I – actualización de código)

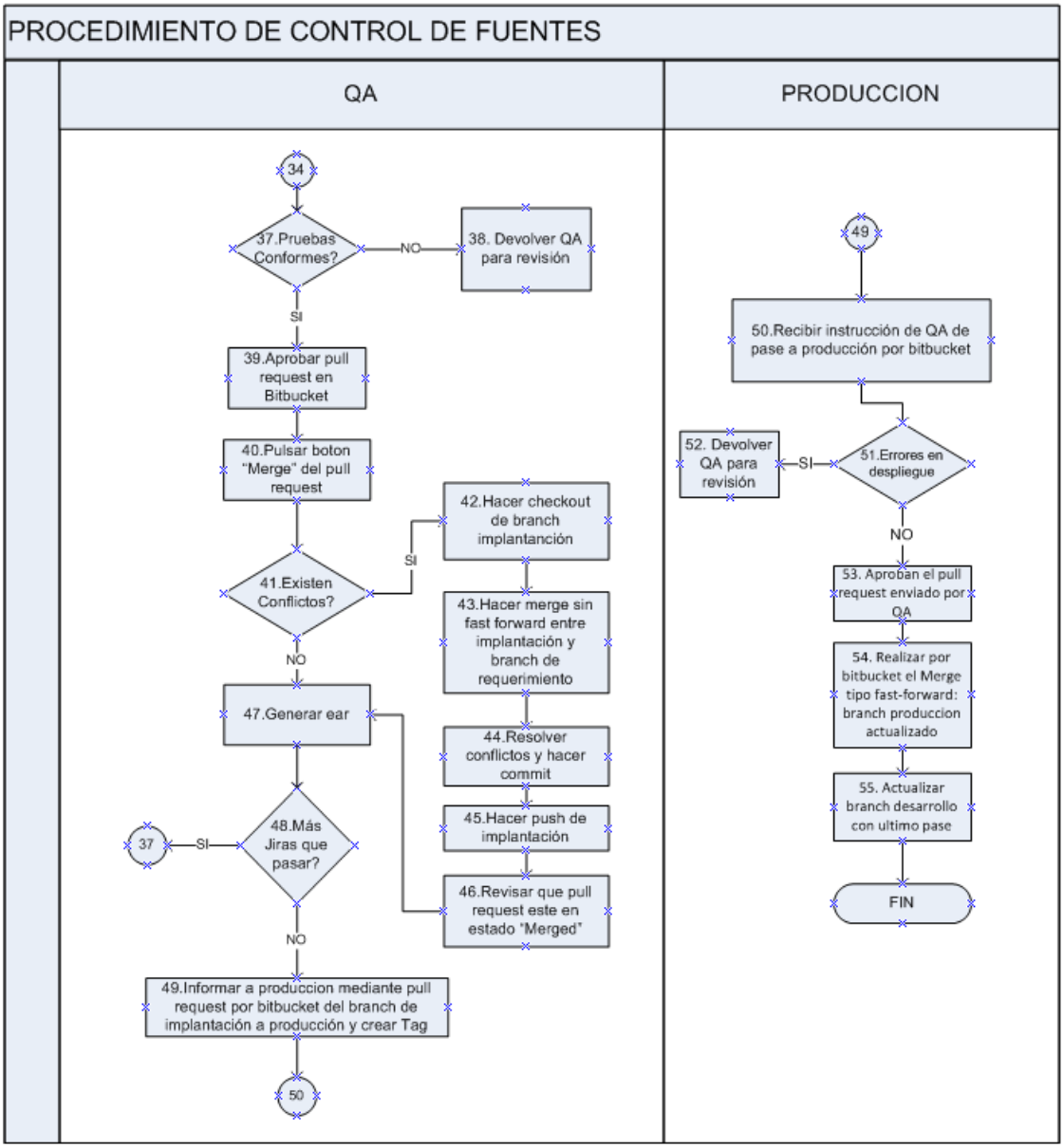


Figura Nº10 (Escenario I – actualización de código)

Escenario II.- Control de fuentes por errores en codificación, pruebas o control de cambio.

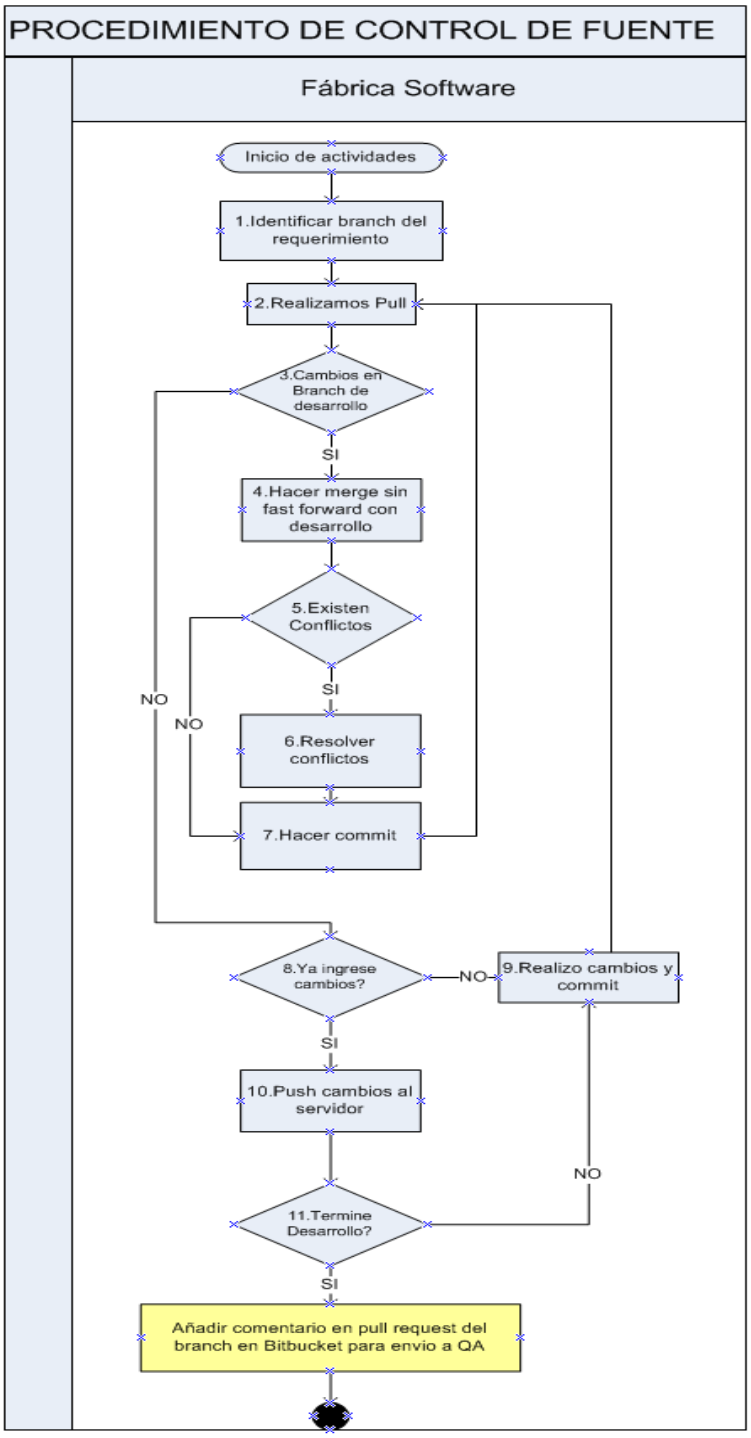


Figura Nº11 (Escenario II – actualización de código)

- *El procedimiento de QA es el mismo realizado en el escenario I, descrito anteriormente.*

Escenario III.- Control de fuentes para un requerimiento o proyecto con más de un desarrollador.

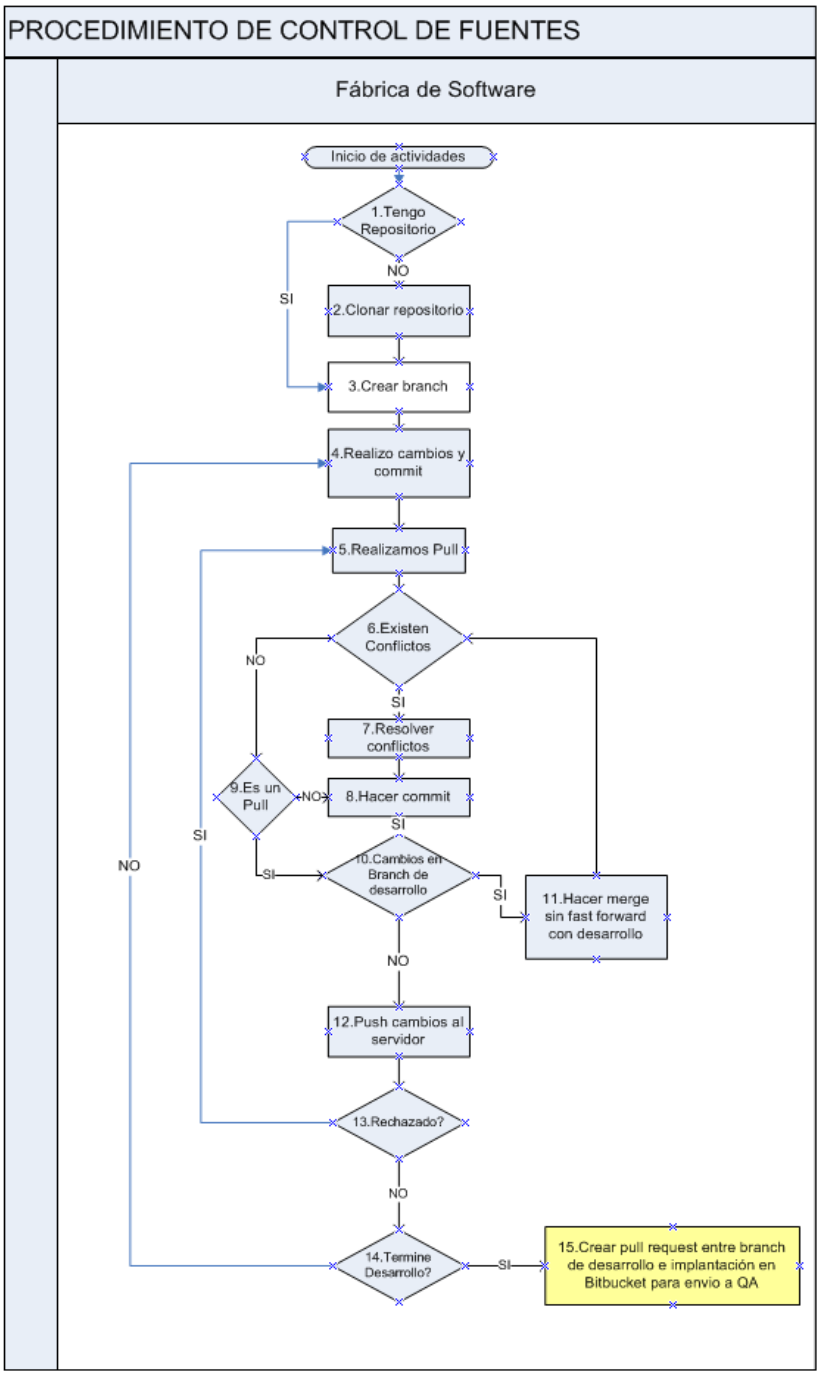


Figura Nº12 (Escenario III – actualización de código)

- *El procedimiento de QA es el mismo realizado en el escenario I, descrito anteriormente*

Escenario IV.- Control de fuentes por errores en codificación, pruebas o control de cambio con más

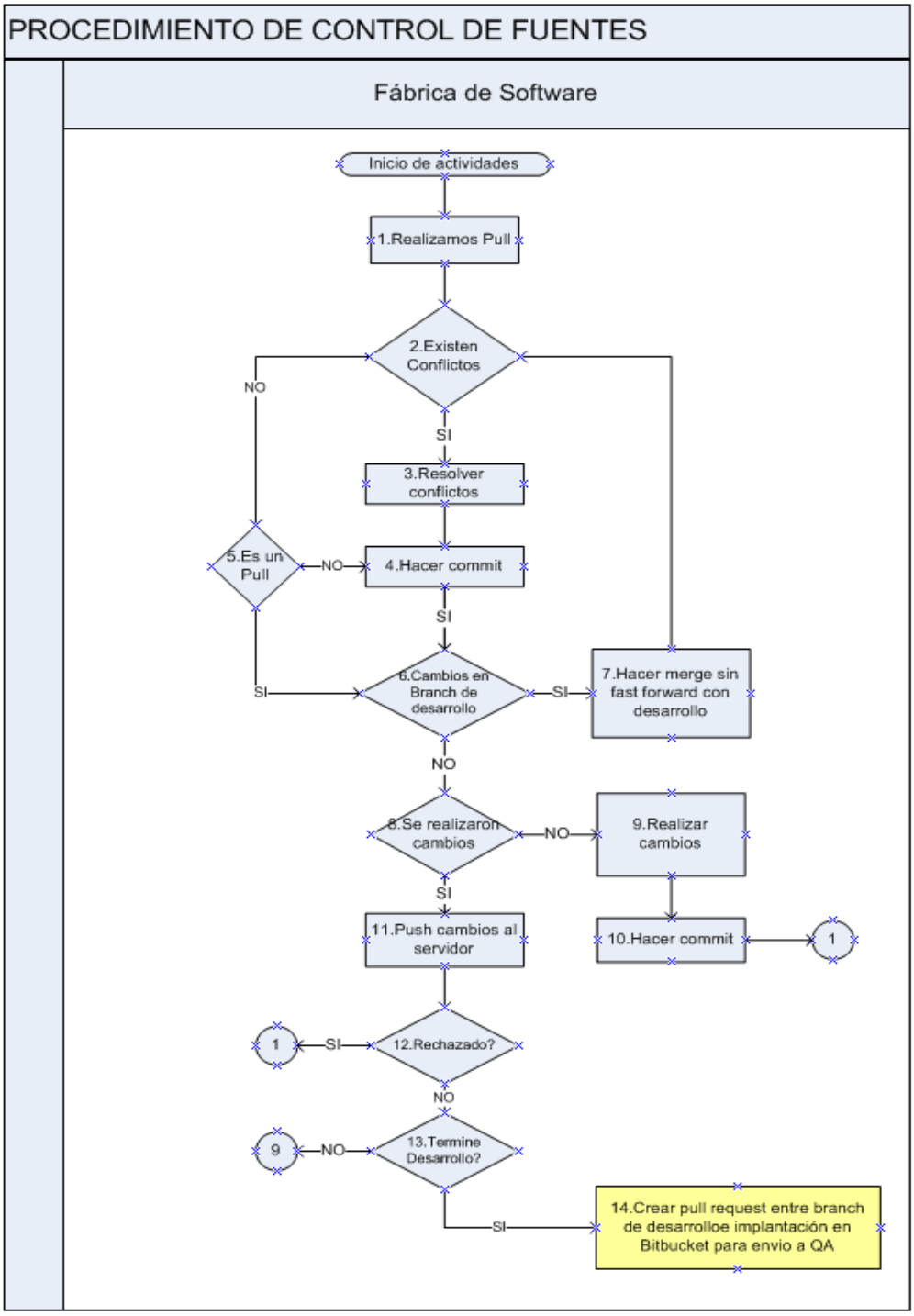


Figura Nº13 (Escenario IV – actualización de código)

- *El procedimiento de QA es el mismo realizado en el escenario I, descrito anteriormente*

- **Narrativa de actividades:**

Las actividades siguientes son utilizadas en uno o mas escenarios, por lo que la numeración es referencial y no se refiere a la indicada en los procedimientos.

N° Act.	Nombre de la actividad	Descripción
1	Clonar repositorio	Se realiza una copia del repositorio que se desea actualizar. Esta actividad puede realizarla la fábrica de software o preproducción según corresponda.
2	Crear branch	Creación de un branch nuevo basado en un repositorio existente, el branch debe tener el nombre del requerimiento jira que se está trabajando. Esta actividad la realiza la fábrica de software
3	Realizo cambios y commits	Realizar cambios: Realiza los cambios en el código de acuerdo lo solicitado en el requerimiento Commit: Grabar los cambios realizados a nivel local en el branch creado Estas actividades son desarrolladas por la fábrica de software
4	Realizamos Pull	Realiza un Fecth al repositorio remoto (actualiza versiones) y commit de no encontrar conflictos a nivel de código.
5	Merge sin fast forward	Se realiza una comparación entre branch local y branch del servidor, identificando las diferencias para un posterior commit
6	Resolver conflictos	Los conflictos en código aparecen cuando dos requerimientos modifican el mismo objeto en la misma línea, al presentarse el conflicto se tiene que resolver de forma manual y luego realizar commit sobre el objeto. Esta actividad es realizada por fábrica de software y QA.
7	Hacer commit	Grabar los cambios realizados en el branch local.
8	Push de cambios al servidor	Esta opción permite grabar los cambios realizados en el branch local hacia el servidor.

9	Crear pull request	La fábrica de software ingresa al aplicativo bitbucket, selecciona el repositorio, el branch y se realiza un pull request con el cual se informa a QA que se está enviando un código fuente.
10	Ejecución Sonar	El sonar es una aplicación que revisa estándares de codificación, esta acción se realizará al momento de recibir el código a QA y se realizará sobre el branch entregado por la fábrica de software.
11	Generar ear de cambio	Generación de ear para su despliegue en preproducción o pase a producción. Esta actividad es realizada por QA.
12	Aprobar pull request en bitbucket	Permite que se pueda integrar el cambio enviado por desarrollo al branch de pase a producción.
13	Pulsar botón Merge de pull request	Realizar Merge entre el branch de desarrollo y pase de producción.
14	Check out al branch	Descargar el código del servidor remoto al branch local.
15	Despliegue conforme	Despliegue en el servidor de preproducción sin problemas.
16	Merge tipo fast-forward	Se realiza una comparación entre branch de origen y branch del destino generándose un nuevo commit de pase a producción.

Políticas

Con el nuevo procedimiento de recepción de código fuente se han elaborado las siguientes políticas:

Actividad	Descripción de las políticas
Recepción de código fuente	• El código será aceptado por QA solo si es enviado mediante bitbucket. • El código será aceptado solo si se trabaja sobre una versión actualizada de producción. • El código será actualizado por control de cambio o error en pruebas sobre la versión inicial entregada que debe estar debidamente actualizada con los pases a producción realizados. • El código pasado a producción será informado a las fábricas de software en el momento de realizado el pase.

Objetivo Específico Nº2:

Elaborar plantillas de estándar de calidad que permita estandarizar el código fuente, detectar a tiempo y reducir errores de desarrollo de software.

En la empresa no se cuenta con plantillas de estándar de calidad de código fuente que permita ser integrado a la herramienta SonarQube ni Jenkins, los escaneos de estándares de calidad se realizan de manera manual lo que conlleva a que se filtren errores los cuales son detectados en etapas más avanzadas ocasionando pérdida de tiempo para la solución de errores y costos dado a las horas consumidas por el equipo de gestión de configuración.

Con el apoyo del equipo de QA y desarrollo de la organización quienes son los usuarios finales de la solución a desarrollar se ha procedido a elaborar las plantillas de calidad el cual será integrado a las herramientas de integración continua Jenkins y SonarQube.

Las siguientes plantillas de estándar de calidad de código fuente han sido elaborados para todos las aplicaciones de la empresa.

- **Plantilla de estándar de calidad de código fuente – WEB**

Nombre de Regla	Tipo	Severidad	Etiquetas
"<frame>" should have a "title"	Mantenibilidad	Mayor	accessibility
"title" should be present in all pages	Mantenibilidad	Mayor	user-experience
Lines should not be too long	Mantenibilidad	Menor	convention
All HTML tags should be closed	Mantenibilidad	Menor	
A <!DOCTYPE> declaration should appear before the <html> tag	Mantenibilidad	Mayor	user-experience
Files should not have too many lines	Mantenibilidad	Mayor	brain-overload
Dynamic includes should not be used	Mantenibilidad	Critico	jsp-jsf
Mouse events should have a corresponding keyboard event	Mantenibilidad	Mayor	accessibility
Images tags and buttons should have an "alt" attribute	Mantenibilidad	Mayor	accessibility
The "style" attribute should not be used	Mantenibilidad	Menor	
Multiple "page" directives should not be used	Mantenibilidad	Menor	convention,jsp-jsf
"autocomplete" should be set to "off" on input elements of type "password"	Seguridad	Critico	html5
Web pages should not contain absolute URIs	Mantenibilidad	Critico	pitfall
JSP expressions should not be used	Mantenibilidad	Mayor	jsp-jsf,obsolete
Flash animations should be embedded using the window mode	Mantenibilidad	Mayor	accessibility
"fieldset" tags should contain a "legend"	Mantenibilidad	Mayor	accessibility

- **Plantilla de estándar de calidad de código fuente – Java**

La siguiente plantilla de estándar de calidad de código fuente es para el código Java.

- **Plantilla de estándar de calidad de código – Javascr**

La siguiente plantilla de estándar de calidad de código fuente es para el código Javascript del producto.

Nombre de Regla	Tipo	Severidad	Etiquetas
"eval" and "arguments" should not be bound or assigned	Confiabilidad	Critico	
JavaScript parser failure	Mantenibilidad	Mayor	
"FIXME" tags should be handled	Mantenibilidad	Mayor	
"TODO" tags should be handled	Mantenibilidad	Informacion	
Lines should not be too long	Mantenibilidad	Menor	convention
Variables should not be self-assigned	Confiabilidad	Mayor	cert
Console logging should not be used	Seguridad	Mayor	owasp-a6
"delete" should not be used on arrays	Confiabilidad	Critico	
Local storage should not be used	Seguridad	Critico	owasp-a6
Non-empty statements should have at least one side-effect	Confiabilidad	Critico	cwe,unused
"new" operators should be used with functions	Confiabilidad	Critico	
"strict" mode should be used with caution	Mantenibilidad	Informacion	cross-browser,user-experience
Collapsible "if" statements should be merged	Mantenibilidad	Mayor	clumsy
"NaN" should not be used in comparisons	Confiabilidad	Bloqueante	

Nombre de Regla	Tipo	Severidad	Etiquetas
Copyright and license headers should be defined	Mantenibilidad	Bloqueante	
Classes and enums with private members should have a constructor	Mantenibilidad	Mayor	pitfall
"Exception" should not be caught when not required by called methods	Seguridad	Critico	cwe,error-handling
Statements should be on separate lines	Mantenibilidad	Menor	convention
Strings literals should be placed on the left side when checking for equality	Mantenibilidad	Mayor	bad-practice
Untrusted data should not be stored in sessions	Seguridad	Critico	cwe
"SingleConnectionFactory" instances should be set to "reconnectOnException"	Confiabilidad	Critico	spring
Control structures should use curly braces	Mantenibilidad	Mayor	cert,misra,pitfall
Exit methods should not be called	Mantenibilidad	Mayor	cwe,suspicious
Throwable.printStackTrace(...) should not be called	Seguridad	Critico	error-handling
Boolean checks should not be inverted	Mantenibilidad	Menor	pitfall
IllegalMonitorStateException should not be caught	Confiabilidad	Critico	multi-threading
Methods should not have too many return statements	Mantenibilidad	Mayor	brain-overload
A "while" loop should be used instead of a "for" loop	Mantenibilidad	Menor	clumsy
"BigDecimal(double)" should not be used	Confiabilidad	Critico	cert
"URL.hashCode" and "URL.equals" should be avoided	Mantenibilidad	Mayor	performance
Try-catch blocks should not be nested	Mantenibilidad	Mayor	confusing
Variables should not be declared before they are relevant	Mantenibilidad	Mayor	brain-overload
Classes and methods that rely on the default system encoding should not be used	Mantenibilidad	Mayor	unpredictable
Fields in a "Serializable" class should either be transient or serializable	Confiabilidad	Critico	cwe,serialization
Invalid "Date" values should not be used	Confiabilidad	Critico	
Non-serializable classes should not be written	Confiabilidad	Critico	serialization

Increment (++) and decrement (--) operators should not be mixed with other operators in an expression	Mantenibilidad	Mayor	cert,misra
Generic exceptions should never be thrown	Seguridad	Critico	cwe,error-handling
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	Mantenibilidad	Mayor	multi-threading,performance
Collections.emptyList(), emptyMap() and emptySet() should be used instead of Collections.EMPTY_LIST, EMPTY_MAP and EMPTY_SET	Mantenibilidad	Mayor	obsolete,pitfall
Package declaration should match source file directory	Mantenibilidad	Mayor	pitfall
Extensions and implementations should not be redundant	Mantenibilidad	Menor	clumsy
Return values should not be ignored when function calls don't have any side effects	Confiabilidad	Critico	cert,misra
".equals()" should not be used to test the values of "Atomic" classes	Confiabilidad	Bloqueante	
"private" methods that don't access instance data should be "static"	Mantenibilidad	Menor	pitfall
Non-serializable objects should not be stored in "HttpSessions"	Confiabilidad	Critico	cwe
"Lock" objects should not be "synchronized"	Mantenibilidad	Mayor	clumsy,multi-threading
Lazy initialization of "static" fields should be "synchronized"	Confiabilidad	Critico	multi-threading
Blocks synchronized on fields should not contain assignments of new objects to those fields	Confiabilidad	Bloqueante	multi-threading
"notifyAll" should be used	Confiabilidad	Critico	multi-threading
Cycles between packages should be removed	Mantenibilidad	Mayor	design
"NOSONAR" should not be used to switch off issues	Mantenibilidad	Mayor	bad-practice
Utility classes should not have public constructors	Mantenibilidad	Mayor	design
Wildcard imports should not be used	Mantenibilidad	Mayor	pitfall
"static" members should be accessed statically	Mantenibilidad	Mayor	pitfall

Unused type parameters should be removed	Mantenibilidad	Mayor	unused
Redundant modifiers should not be used	Mantenibilidad	Menor	clumsy
Instance methods should not write to "static" fields	Confiabilidad	Critico	multi-threading
"indexOf" checks should not be for positive numbers	Mantenibilidad	Critico	pitfall
Threads should not be started in constructors	Mantenibilidad	Mayor	multi-threading,pitfall
Inner classes which do not reference their owning classes should be "static"	Mantenibilidad	Mayor	performance
"PreparedStatement" and "ResultSet" methods should be called with valid indices	Confiabilidad	Bloqueante	sql
Artifact ids should follow a naming convention	Mantenibilidad	Menor	convention,maven
pom elements should be in the recommended order	Mantenibilidad	Menor	convention,maven
"@Deprecated" code should not be used	Mantenibilidad	Menor	cwe,obsolete
"deleteOnExit" should not be used	Mantenibilidad	Mayor	performance
Unused local variables should be removed	Mantenibilidad	Mayor	unused
"InterruptedException" should not be ignored	Confiabilidad	Critico	cwe,multi-threading
Catches should be combined	Mantenibilidad	Mayor	clumsy
Mutable fields should not be "public static"	Seguridad	Critico	cwe,unpredictable
Child class members should not shadow parent class members	Mantenibilidad	Mayor	confusing
Constructor injection should be used instead of field injection	Mantenibilidad	Mayor	design,pitfall
The signature of "finalize()" should match that of "Object.finalize()"	Mantenibilidad	Mayor	pitfall
Dead stores should be removed	Mantenibilidad	Mayor	cert,cwe,suspicious,unused
"toString()" should never be called on a String object	Mantenibilidad	Mayor	clumsy,pitfall
Mutable members should not be stored or returned directly	Seguridad	Critico	cert,cwe,unpredictable
Public constants and fields initialized at declaration should	Mantenibilidad	Menor	convention

be "static final" rather than merely "final"			
Methods of "Random" that return floating point values should not be used in random integer generation	Mantenibilidad	Mayor	clumsy
Collapsible "if" statements should be merged	Mantenibilidad	Mayor	clumsy
Expressions should not be too complex	Mantenibilidad	Mayor	brain-overload
Synchronization should not be based on Strings or boxed primitives	Confiabilidad	Bloqueante	cert,multi-threading
Member variable visibility should be specified	Seguridad	Critico	
"final" classes should not have "protected" members	Mantenibilidad	Mayor	confusing
"Cloneables" should implement "clone"	Confiabilidad	Critico	
Silly equality checks should not be made	Confiabilidad	Critico	unused
Methods should not be empty	Mantenibilidad	Mayor	suspicious
Exceptions should not be thrown from servlet methods	Seguridad	Critico	cert,cwe,error-handling,owasp-a6
"DateUtils.truncate" from Apache Commons Lang library should not be used	Mantenibilidad	Mayor	java8,performance
Static non-final field names should comply with a naming convention	Mantenibilidad	Menor	convention
Public types, methods and fields (API) should be documented with Javadoc	Mantenibilidad	Menor	convention
Local variables should not shadow class fields	Mantenibilidad	Mayor	pitfall
Throwable and Error should not be caught	Seguridad	Bloqueante	cert,cwe,error-handling
"ScheduledThreadPoolExecutor" should not have 0 core threads	Confiabilidad	Bloqueante	
"runFinalizersOnExit" should not be called	Confiabilidad	Bloqueante	cert
Labels should not be used	Mantenibilidad	Mayor	confusing
Values should not be uselessly incremented	Confiabilidad	Critico	
Comma operator should not be used	Mantenibilidad	Mayor	misra

Constructors should not be used to instantiate "String" and primitive-wrapper classes	Mantenibilidad	Mayor	performance
Null pointers should not be dereferenced	Confiabilidad	Bloqueante	cert,cwe,owasp-a1,owasp-a2,owasp-a6
"assert" should only be used with boolean variables	Mantenibilidad	Mayor	suspicious
Exception classes should be immutable	Mantenibilidad	Mayor	error-handling
Exception handlers should preserve the original exception	Mantenibilidad	Critico	error-handling
"HttpServletRequest.getRequestedSessionId()" should not be used	Seguridad	Critico	cwe,owasp-a2,sans-top25-porous
Only standard cryptographic algorithms should be used	Seguridad	Bloqueante	cwe,owasp-a6,sans-top25-porous
Long suffix "L" should be upper case	Mantenibilidad	Menor	
Methods and field names should not be the same or differ only by capitalization	Mantenibilidad	Mayor	confusing
Los objetos no deben crearse para ser eliminados inmediatamente sin ser utilizados	Confiabilidad	Critico	unused
"Iterator.hasNext()" should not call "Iterator.next()"	Confiabilidad	Bloqueante	
Los métodos públicos deberían arrojar como máximo una excepción marcada	Mantenibilidad	Mayor	error-handling
Los bucles no deben contener más de una declaración de "interrupción" o "continuar"	Mantenibilidad	Mayor	brain-overload
El análisis se debe usar para convertir "Cadenas" en primitivas	Mantenibilidad	Mayor	performance
"ConcurrentLinkedQueue.size()" should not be used	Seguridad	Critico	performance
Una cláusula de actualización de bucle "para" debe mover el contador en la dirección correcta	Confiabilidad	Bloqueante	

"entrySet()" should be iterated when both the key and value are needed	Mantenibilidad	Mayor	performance
Annotation - arguments should appear in the order - in which they were declared	Mantenibilidad	Menor	convention
Literal suffixes should be upper case	Mantenibilidad	Menor	cert,convention,misra,pitfall
Las asignaciones no deben hacerse desde dentro de las sub-expresiones	Confiabilidad	Mayor	cwe,misra
Los campos en clases no serializables no deben ser "transitorios"	Mantenibilidad	Menor	serialization,unused
Credentials should not be hard-coded	Seguridad	Critico	cwe,owasp-a2,sans-top25-porous
Loops should not be infinite	Confiabilidad	Bloqueante	cert
Raw byte values should not be used in bitwise operations in combination with shifts	Confiabilidad	Critico	
Abstract methods should not be redundant	Mantenibilidad	Mayor	confusing
"private" methods called only by inner classes should be moved to those classes	Mantenibilidad	Mayor	confusing
El método Object.finalize () no se debe llamar	Seguridad	Critico	cert,cwe
Java parser failure	Mantenibilidad	Mayor	suspicious
Variables should not be self-assigned	Confiabilidad	Critico	cert
Multiple variables should not be declared on the same line	Mantenibilidad	Menor	convention
Comparators should be "Serializable"	Mantenibilidad	Mayor	pitfall,serialization
Las clases internas "serializables" de clases no serializables deben ser "estáticas"	Confiabilidad	Critico	serialization
Math operands should be cast before assignment	Confiabilidad	Critico	cwe,sans-top25-risky
Silly math should not be performed	Mantenibilidad	Mayor	clumsy

Standard outputs should not be used directly to log anything	Mantenibilidad	Mayor	bad-practice
Magic numbers should not be used	Mantenibilidad	Menor	brain-overload
"Externalizable" classes should have a no-arguments constructor	Confiabilidad	Mayor	
Las firmas de métodos de serialización personalizados deben cumplir los requisitos	Confiabilidad	Critico	
"readResolve" methods should be inheritable	Mantenibilidad	Mayor	pitfall
Los valores pasados a los comandos del sistema operativo deben ser desinfectados	Seguridad	Critico	cwe,owasp-a1,sans-top25-insecure
Multiple loops over the same set should be combined	Mantenibilidad	Mayor	performance
Empty statements should be removed	Mantenibilidad	Menor	cert,misra,unused
Classes should not be compared by name	Confiabilidad	Critico	cwe
"static final" arrays should be "private"	Seguridad	Critico	cwe
"for" loop incrementers should modify the variable being tested in the loop's stop condition	Mantenibilidad	Mayor	suspicious
"@NonNull" values should not be set to null	Confiabilidad	Critico	misra
Method overrides should not change contracts	Mantenibilidad	Mayor	suspicious
Inappropriate regular expressions should not be used	Confiabilidad	Critico	
"java.lang.Error" should not be extended	Seguridad	Menor	error-handling
Files should contain only one top-level class or interface each	Mantenibilidad	Mayor	brain-overload
"finalize" should not set fields to "null"	Mantenibilidad	Mayor	clumsy,performance
"compareTo" should not return "Integer.MIN_VALUE"	Confiabilidad	Critico	
"action" mappings should not have too many "forward" entries	Mantenibilidad	Mayor	brain-overload,struts
Struts validation forms should have unique names	Confiabilidad	Critico	cwe,struts

Unused protected methods should be removed	Mantenibilidad	Mayor	unused
String literals should not be duplicated	Mantenibilidad	Menor	design
Maps with keys that are enum values should be replaced with EnumMap	Mantenibilidad	Mayor	performance
Sets with elements that are enum values should be replaced with EnumSet	Mantenibilidad	Mayor	performance
Las cadenas no deben concatenarse usando '+' en un bucle	Mantenibilidad	Mayor	performance
Las expresiones idénticas no se deben usar en ambos lados de un operador binario	Confiabilidad	Critico	cert
Las clases internas "Serializables" de las clases "Serializables" deben ser estáticas	Mantenibilidad	Mayor	pitfall,serialization
Child class methods named for parent class methods should be overrides	Mantenibilidad	Mayor	pitfall
Classes without "public" constructors should be "final"	Mantenibilidad	Mayor	design
String function use should be optimized for single characters	Mantenibilidad	Menor	clumsy,performance
Los campos variables de clase no deben tener acceso público	Mantenibilidad	Mayor	cwe
se debe llamar a super.finalize () al final de las implementaciones Object.finalize ()	Confiabilidad	Bloqueante	cert,cwe
The non-serializable super class of a "Serializable" class should have a no-argument constructor	Confiabilidad	Critico	serialization
"Serializable" classes should have a version id	Mantenibilidad	Mayor	pitfall,serialization
Inappropriate "Collection" calls should not be made	Confiabilidad	Critico	
Los nombres de clase no deben sombrear interfaces o superclases	Mantenibilidad	Mayor	pitfall

La lógica de cortocircuito debería usarse en contextos booleanos	Confiabilidad	Critico	
The diamond operator ("<>") should be used	Mantenibilidad	Mayor	clumsy
"catch" clauses should do more than rethrow	Mantenibilidad	Mayor	clumsy,unused
"clone" should not be overridden	Mantenibilidad	Mayor	suspicious
"File.createTempFile" should not be used to create a directory	Seguridad	Critico	owasp-a9
Files should not be empty	Mantenibilidad	Mayor	unused
Throws declarations should not be superfluous	Mantenibilidad	Menor	clumsy,unused
"equals (Object obj)" debe anularse junto con el método "compareTo (T obj)"	Confiabilidad	Critico	
Los miembros de una declaración o clase de interfaz deben aparecer en un orden predefinido	Mantenibilidad	Menor	convention
Constants should not be defined in interfaces	Mantenibilidad	Menor	bad-practice
Los tipos comodín genéricos no se deben usar en los parámetros de devolución	Mantenibilidad	Mayor	pitfall
Una clase abstracta debe tener métodos abstractos y concretos	Mantenibilidad	Menor	convention
"NullPointerException" should not be explicitly thrown	Mantenibilidad	Mayor	pitfall
"NullPointerException" should not be caught	Mantenibilidad	Mayor	cert,cwe,error-handling
La lógica de cortocircuito se debe usar para evitar las desreferencias del puntero nulo en los condicionales	Confiabilidad	Bloqueante	
Los objetos se deben comparar con "iguales ()"	Mantenibilidad	Mayor	cert,cwe
Nested "enum"s should not be declared static	Mantenibilidad	Mayor	clumsy

Thread.run () y Runnable.run () no deberían invocarse directamente	Confiabilidad	Critico	cert,cwe,multi-threading
las declaraciones de "cambio" no deben contener etiquetas que no sean de casos	Mantenibilidad	Mayor	misra,suspicious
Los métodos no constructor no deberían tener el mismo nombre que la clase adjunta	Mantenibilidad	Mayor	pitfall
Constructors should only call non-overridable methods	Mantenibilidad	Mayor	pitfall
The value returned from a stream read should be checked	Confiabilidad	Bloqueante	
Neither "Math.abs" nor negation should be used on numbers that could be "MIN_VALUE"	Confiabilidad	Critico	
"read" and "readLine" return values should be used	Confiabilidad	Bloqueante	
Methods should not return constants	Mantenibilidad	Mayor	confusing
"for" loop stop conditions should be invariant	Mantenibilidad	Mayor	misra,pitfall
IP addresses should not be hardcoded	Seguridad	Critico	cert
Las operaciones de poco tonto no deberían realizarse	Mantenibilidad	Mayor	suspicious
"Threads" no deberían usarse donde se espera que se ejecuten "Runnables"	Mantenibilidad	Mayor	multi-threading,pitfall
"readObject" should not be "synchronized"	Mantenibilidad	Mayor	confusing
"Calendars" and "DateFormats" should not be static	Confiabilidad	Critico	multi-threading
El árbol de clases de herencia no debe ser demasiado profundo	Mantenibilidad	Mayor	design
Loggers should be "private static final" and should share a naming convention	Mantenibilidad	Menor	convention
Octal values should not be used	Mantenibilidad	Mayor	cert,misra,pitfall

HTTP referers should not be relied on	Seguridad	Critico	cwe,owasp-a2,sans-top25-porous
Fields should not be initialized to default values	Mantenibilidad	Menor	convention
Methods named "equals" should override Object.equals(Object)	Mantenibilidad	Mayor	suspicious
"public static" fields should be constant	Seguridad	Critico	cert,cwe
Web applications should not have a "main" method	Seguridad	Critico	cwe,jee
"enum" fields should not be publicly mutable	Seguridad	Critico	bad-practice
Strings should be compared using "equals()"	Mantenibilidad	Critico	cwe
Locale should be used in String operations	Mantenibilidad	Mayor	cert,unpredictable
Classes should not be loaded dynamically	Seguridad	Critico	cwe,owasp-a1
Cookies should be "secure"	Seguridad	Critico	cwe,owasp-a2,owasp-a6
Classes should not be empty	Mantenibilidad	Mayor	clumsy
Resources should be closed	Confiabilidad	Bloqueante	cert,cwe,denial-of-service,leak
"main" should not "throw" anything	Seguridad	Critico	error-handling
"equals(Object obj)" should test argument type	Confiabilidad	Bloqueante	
Try-with-resources should be used	Mantenibilidad	Mayor	pitfall
Source files should not have any duplicated blocks	Mantenibilidad	Mayor	pitfall
Unused function parameters should be removed	Mantenibilidad	Mayor	misra,unused
Debugger statements should not be used	Seguridad	Critico	cwe,user-experience
Function declarations should not be made within blocks	Mantenibilidad	Mayor	cross-browser,user-experience
Octal values should not be used	Mantenibilidad	Mayor	cert,misra,pitfall
Built-in objects should not be overridden	Confiabilidad	Critico	confusing
"arguments" should not be accessed directly	Mantenibilidad	Mayor	api-design,es2015

Single quotes should be used for string literals	Mantenibilidad	Menor	convention
Variables should not be shadowed	Mantenibilidad	Mayor	pitfall
"===" and "!==" should be used instead of "==" and "!="	Confiabilidad	Mayor	
HTML-style comments should not be used	Confiabilidad	Mayor	
Only "while", "do" and "for" statements should be labelled	Mantenibilidad	Mayor	pitfall
The base should be provided to "parseInt"	Confiabilidad	Critico	
The "changed" property should not be manipulated directly	Confiabilidad	Critico	backbone
Variables should be declared explicitly	Mantenibilidad	Mayor	pitfall
Control structures should always use curly braces	Mantenibilidad	Mayor	cert,cwe,misra,pitfall
Wrapper objects should not be used for primitive types	Mantenibilidad	Mayor	pitfall
Dead Stores should be removed	Confiabilidad	Mayor	cert,cwe,unused
Related "if/else if" statements and "cases" in a "switch" should not have the same condition	Confiabilidad	Critico	cert,pitfall,unused
Cross-document messaging domains should be carefully restricted	Seguridad	Critico	html5,owasp-a3
The identity operator ("===") should not be used with dissimilar types	Confiabilidad	Critico	
Each statement should end with a semicolon	Mantenibilidad	Menor	convention
Unused local variables should be removed	Mantenibilidad	Mayor	unused
Array and Object constructors should not be used	Confiabilidad	Mayor	
Constructor functions should not be called purely for side-effects	Mantenibilidad	Mayor	pitfall

"continue" should not be used	Mantenibilidad	Critico	misra
Statements should be on separate lines	Mantenibilidad	Menor	convention
Expressions should not be too complex	Mantenibilidad	Mayor	brain-overload
Setters should not return values	Confiabilidad	Critico	
Comparison operators should not be used with strings	Mantenibilidad	Mayor	suspicious
Multiline string literals should not be used	Mantenibilidad	Mayor	bad-practice
Results of operations on strings should not be ignored	Confiabilidad	Bloqueante	
Values should not be uselessly incremented	Confiabilidad	Critico	
Variable declarations should be placed appropriately for their scope	Mantenibilidad	Mayor	pitfall
Element type selectors should not be used with class selectors	Mantenibilidad	Mayor	jquery,performance,user-experience
Universal selectors should not be used	Mantenibilidad	Mayor	jquery,performance,user-experience
Selections should be stored	Mantenibilidad	Mayor	jquery,performance,user-experience
Trailing commas should not be used	Mantenibilidad	Bloqueante	cross-browser
Bitwise operators should not be used	Mantenibilidad	Mayor	pitfall
"if ... else if" constructs shall be terminated with an "else" clause	Mantenibilidad	Mayor	cert,misra
Local variables should not shadow "undefined"	Mantenibilidad	Critico	pitfall
"undefined" should not be assigned	Mantenibilidad	Critico	pitfall
Selection results should be tested with "length"	Confiabilidad	Critico	jquery
Jump statements should not be followed by other statements	Mantenibilidad	Mayor	cert,cwe,misra,unused
"with" statements should not be used	Confiabilidad	Mayor	

"alert(...)" should not be used	Seguridad	Mayor	cwe,user-experience
Deprecated jQuery methods should not be used	Mantenibilidad	Mayor	jquery,obsolete
Variables and functions should not be redeclared	Confiabilidad	Mayor	pitfall
Web SQL databases should not be used	Seguridad	Critico	html5,owasp-a6,owasp-a9
"[type=...]" should be used to select elements by type	Mantenibilidad	Mayor	jquery,performance
Lines should not end with trailing whitespaces	Mantenibilidad	Menor	convention
Functions should not be defined inside loops	Confiabilidad	Mayor	
Function calls should not pass extra arguments	Confiabilidad	Critico	cwe,misra
Source files should not have any duplicated blocks	Mantenibilidad	Mayor	pitfall

Objetivo Específico Nº3

En la empresa para el proceso de desarrollo de software se usaban las siguientes herramientas: Eclipse Índigo, SVN. Las mismas herramientas eran usadas por más de 7 años en la empresa, el proceso de actualización se realizaba de manera manual, los errores de código eran encontrados en etapas muy avanzadas como en la etapa de pruebas de calidad, causando de esta manera atraso en la entrega del producto.

Con el objetivo de estandarizar la calidad del código fuente ha sido necesaria la implantación de las siguientes herramientas de integración continua: Git, SonarQube, Jenkins y Bitbucket, las cuales trabajarán de manera integrada.

- **GIT**

Como parte de la solución se procedió a migrar de SVN a GIT. De tal manera que el problema que tenían los analistas QA y desarrolladores para ingresar a subir cambios o actualizar repositorios locales en una misma PC no sea un atraso en el desarrollo del software, con la implantación de esta herramienta los desarrolladores y analistas QA podrán obtener el código actualizado en cada una de sus PC y subir cambios en el momento que lo requieran sin necesidad de esperar un turno para ingresar a una sóla máquina perdiendo tiempo y ocasionando muchas veces demoras en el inicio de testeo de las aplicaciones.

Con la implantación de la herramienta GIT, ahora el proceso de actualización de código fuente es más amigable y confiable, pues GIT permite actualizar los repositorios locales en un tiempo mucho más corto que cuando se realizaba de manera manual, pues antes se comparaba objeto por objeto validando estándares de programación y muchas veces omitiendo cambios muy significativos enviados por desarrollo.

- **SONARQUBE**

Para configurar el escaneo de sonar de manera automática, primero se debe comprender el proceso manual de actualización de código fuente que sigue el siguiente flujo:

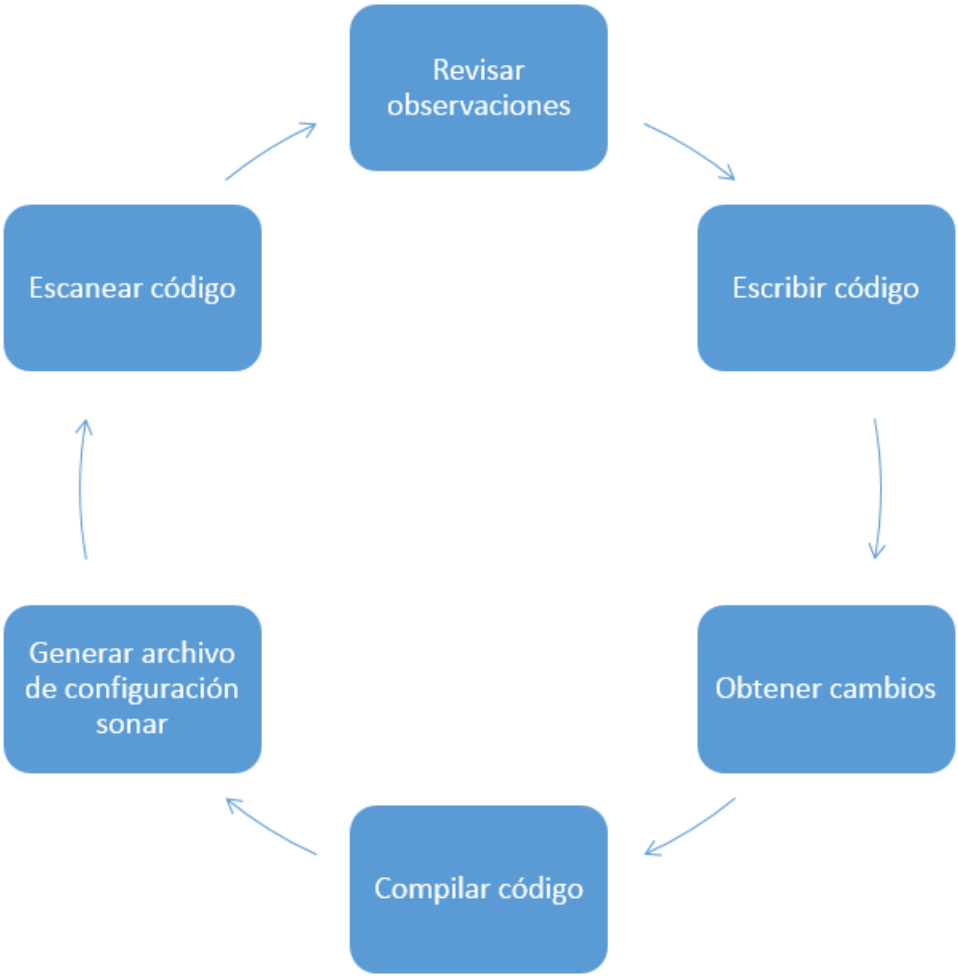


Figura Nº14 (Proceso manual de actualización de código)

Del flujo se identificó que es posible automatizar los siguientes pasos:

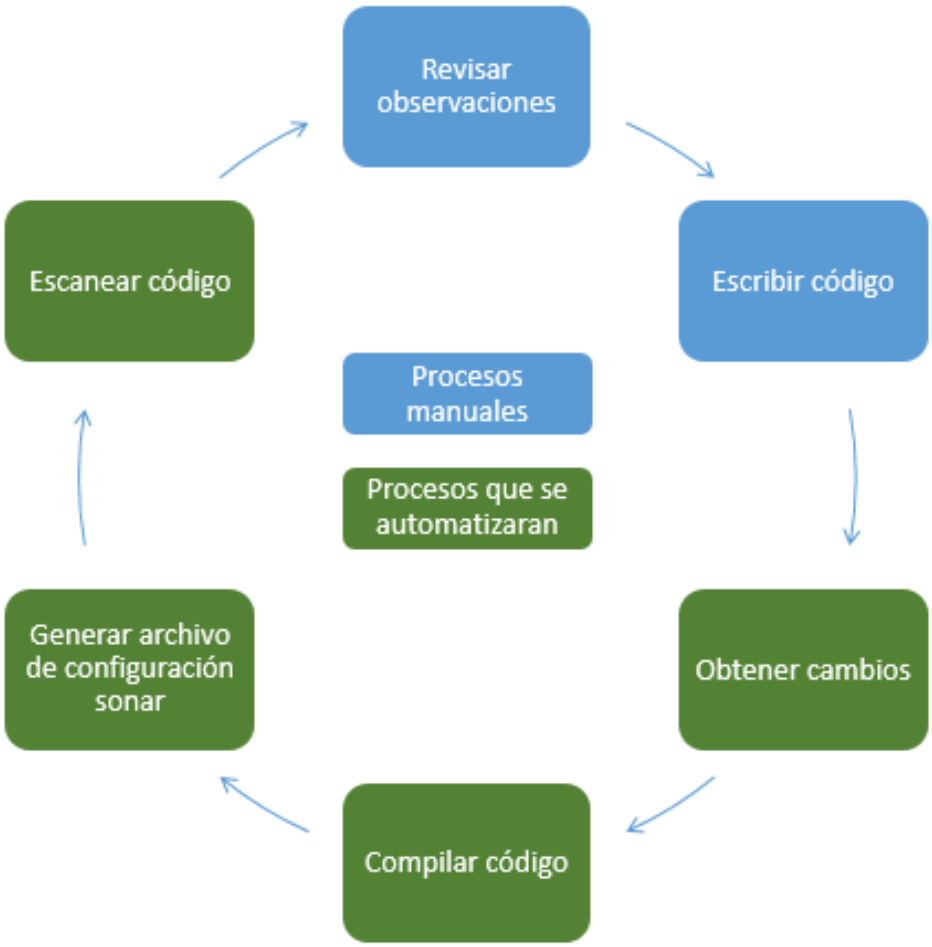


Figura Nº15 (Proceso automatizado de actualización de código)

Dado que Sonar permite gestionar la calidad de código fuente de las aplicaciones, al instalarla el cliente ahora puede analizar, recopilar y visualizar métricas del código fuente, permitiendo tener informes con resúmenes de las métricas realizadas después del escaneo a las aplicaciones.

La aplicación de la herramienta SONAR permite al cliente analizar y seleccionar reglas de programación identificando las mejores prácticas al momento de desarrollar una aplicación, además permite la identificación de errores antes de instalar un cambio en los ambientes de preproducción (QA).

El cliente puede visualizar los tipos de errores de programación del software: Blocker, Critical, Major, Minor e Info.

De acuerdo a lo establecido con el cliente se recomendó que Preproducción no acepte cambios con issues en los siguientes niveles de severidad:

- Blocker
- Critical
- Major

De esta manera el cliente puede elaborar Informes con resúmenes de las métricas realizadas después del escaneo de las aplicaciones.

El cliente puede tomar decisiones para mejora en el filtro de contratación de proveedores de desarrollo de sus aplicaciones, en base a los resultados de las métricas.

- **JENKINS**

Mediante la herramienta Jenkins los analistas QA puede validar los proyectos enviados por desarrollo ejecutando un disparador a la herramienta SONAR quien será el encargado de escanear el código bajo las métricas establecidas en el paso anterior.

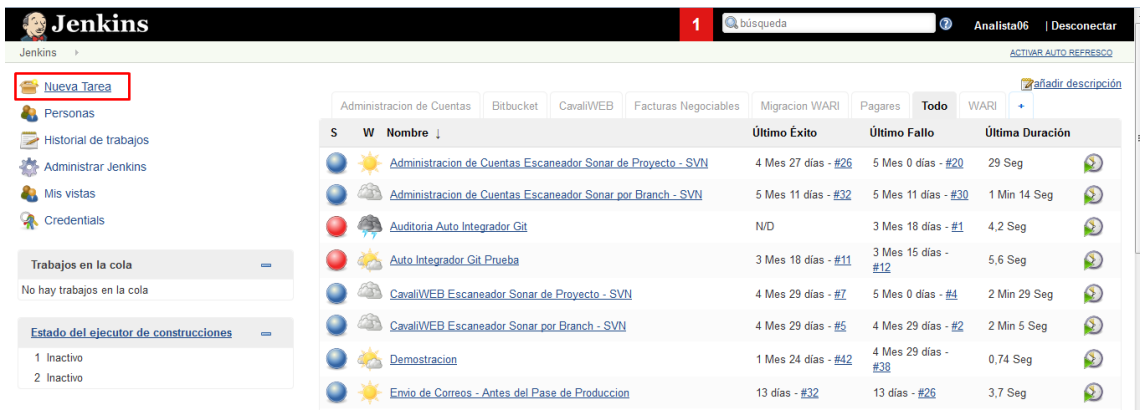


Figura Nº16 (JENKINS)

CAPÍTULO 4

RESULTADOS

4.1 Resultados

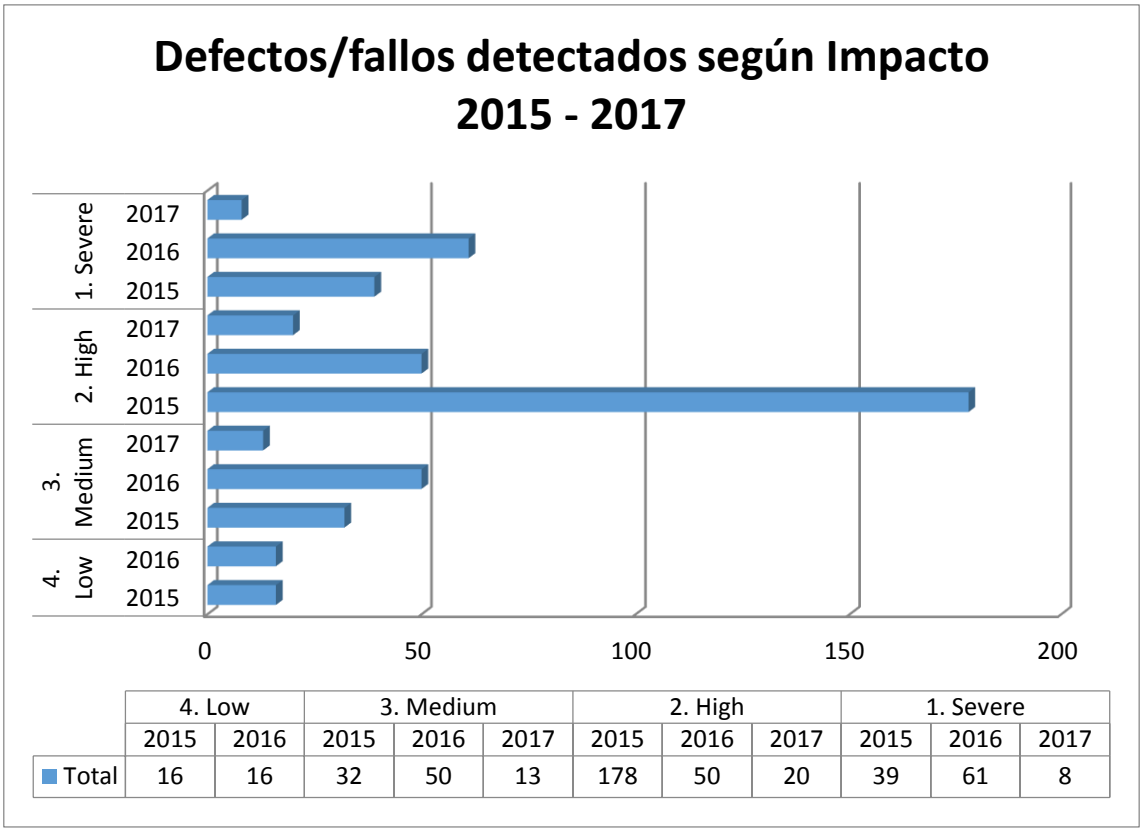
4.1.1 Resultados

Objetivo Especifico 1:

- A través de constantes reuniones y coordinaciones vía correo programadas con los jefes de las areas involucradas en la solución brindada, se logró establecer las herramientas a utilizar para el desarrollo de la solución en base a las necesidades así como posibilidades económicas de la empresa. (Anexo 1, 2 y 3)
- Con la información recopilada (Ver Anexo 4) se logró diseñar un nuevo proceso de actualización de código fuente (Ver anexo 5, 6 y 7), considerando escenarios críticos e ideales en el desarrollo de software.

Objetivo Especifico 2:

- Con la elaboración de las plantillas de estándares de código fuente se logró mejorar la calidad del producto reduciendo la cantidad de errores en el sistema así como el nivel de severidad de los mismos.



	4. Low		3. Medium			2. High			1. Severe		
	2015	2016	2015	2016	2017	2015	2016	2017	2015	2016	2017
Total	16	16	32	50	13	178	50	20	39	61	8

Figura Nº17 (Defectos según impacto 2015-2017)

Cuenta de ISSUE		
AÑO	IMPACT	Total
2015	4. Low	16
	3. Medium	32
	2. High	178
	1. Severe	39
Total general		**265**

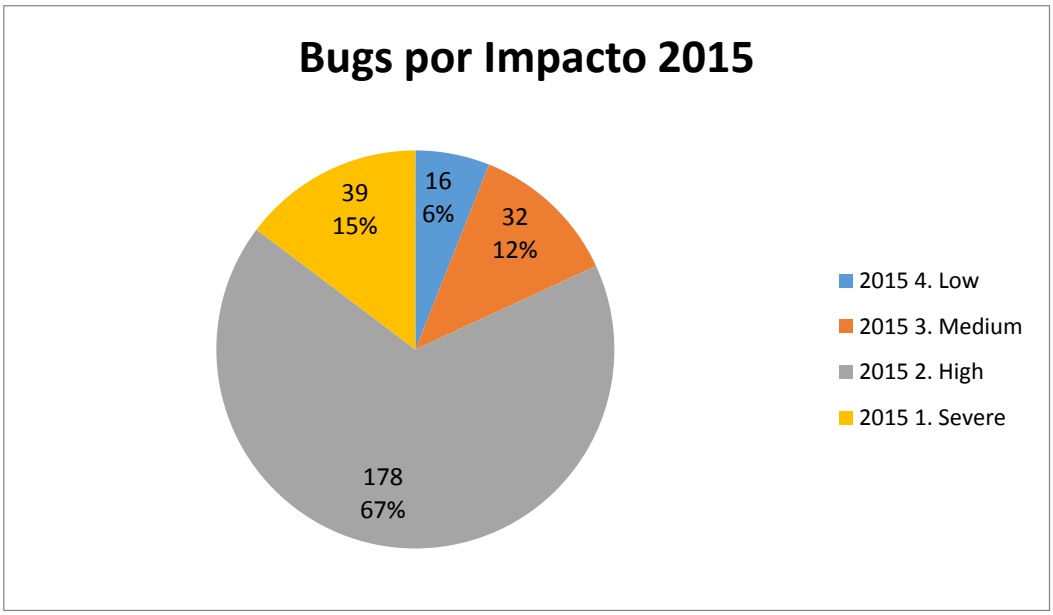


Figura Nº18 (Defectos 2015)

Cuenta de ISSUE		
AÑO	IMPACT	Total
2016	4. Low	16
	3. Medium	50
	2. High	50
	1. Severe	61
Total general		**177**

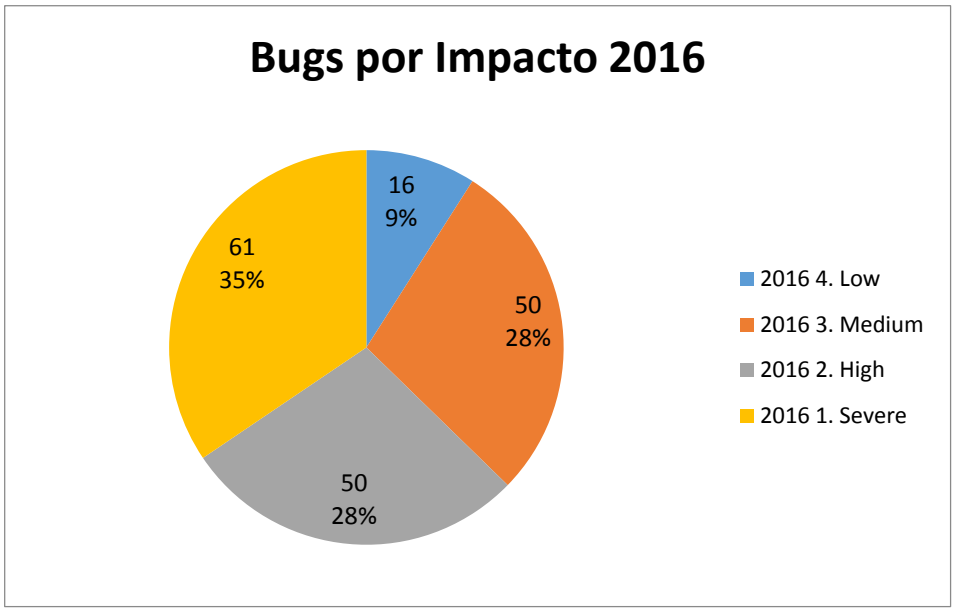


Figura Nº19 (Defectos 2016)

Cuenta de ISSUE		
AÑO	IMPACT	Total
2017	3. Medium	13
	2. High	20
	1. Severe	8
Total general		**41**

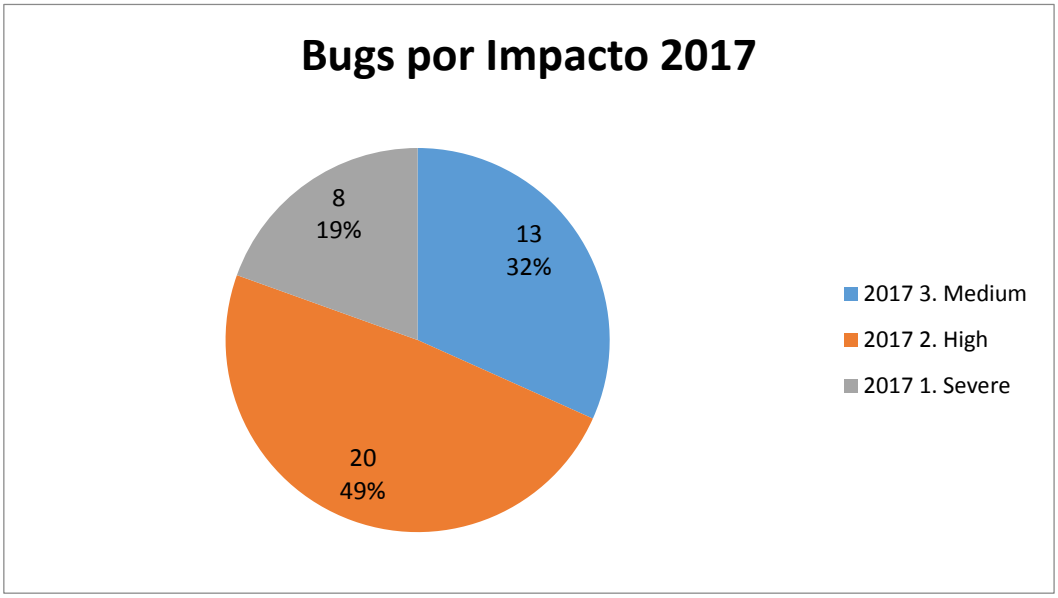


Figura Nº20 (Defectos 2017)

- Los errores ahora son detectados en etapas tempranas del desarrollo de software, permitiendo a los programadores corregirlos con mayor facilidad evitando pérdida de tiempo por las constantes devoluciones del área de calidad.

Cuenta de ISSUE		IMPACT				
AÑO	MES	4. Low	3. Medium	2. High	1. Severe	Total general
2015	9	16	9	35	17	77
	10		16	70		86
	11		4	63	21	88
	12		3	10	1	14
2016	1	1	7	3	2	13
	2		2	3		5
	3		3	2		5
	4		19	6		25
	5		2	5		7
	6		1	2		3
	10	7	12	20	42	81
	11	4	1	3	10	18
	12	4	3	6	7	20
2017	1		3	7	2	12
	2				2	2
	3		8	4	1	13
	4		2	7	3	12
	5			2		2

Total general		**32**	**95**	**248**	**108**	**483**

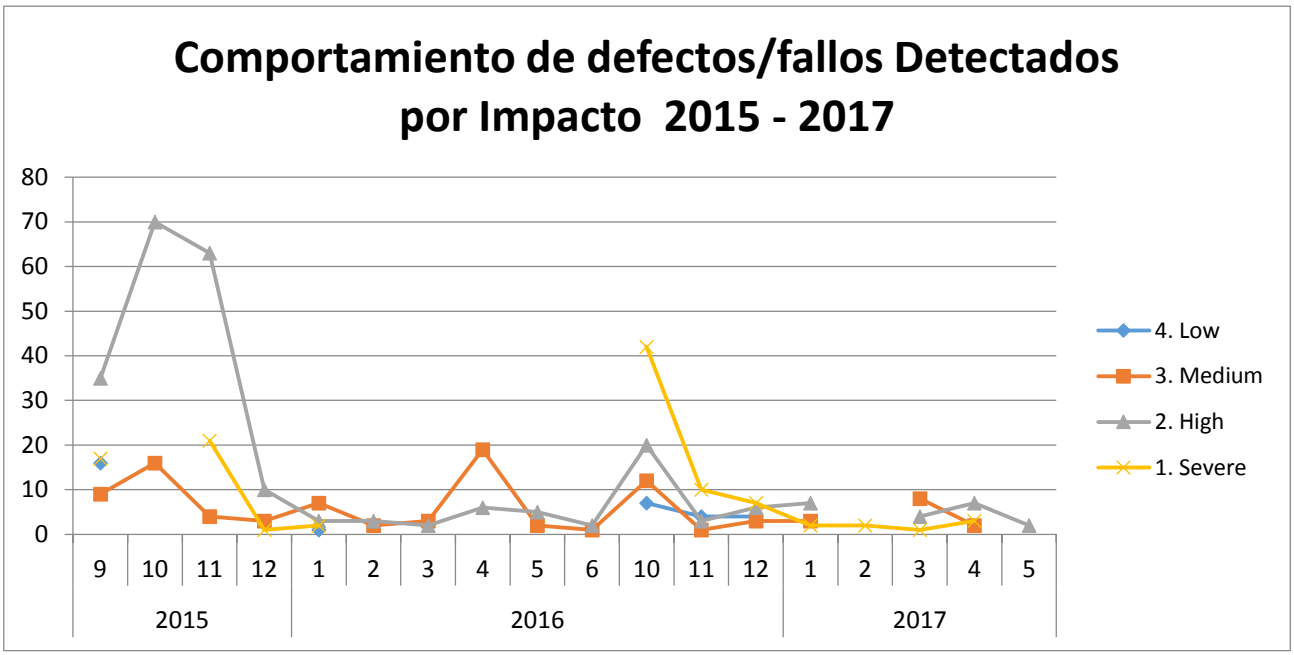


Figura Nº21 (Comportamiento de Defectos detectados)

Objetivo Especifico 3:

- Se logró estandarizar la calidad de código fuente con la implantación de las herramientas SonarQube y Jenkins reduciendo los errores de desarrollo y ayudando al cliente a identificar la calidad del servicio de sus proveedores de desarrollo.

- Se ha evitado el gasto de la empresa para la corrección de errores funcionales futuros, de acuerdo a los indicadores de Sonar se puede deducir la cantidad de tiempo en la corrección de 111 días aproximadamente.

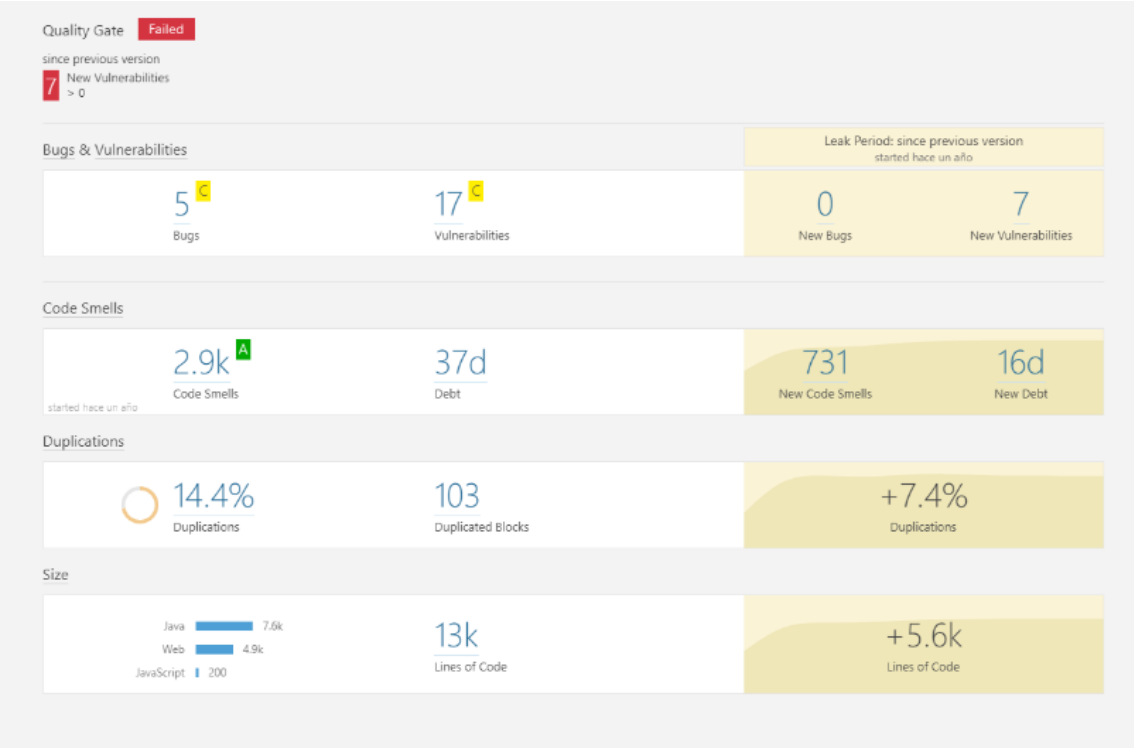


Figura Nº22 (Escaneo de proyecto con fallas)

Como se muestra en la siguiente imagen, la cantidad de bugs y vulnerabilidades para un aplicativo específico se ha reducido al 100%:

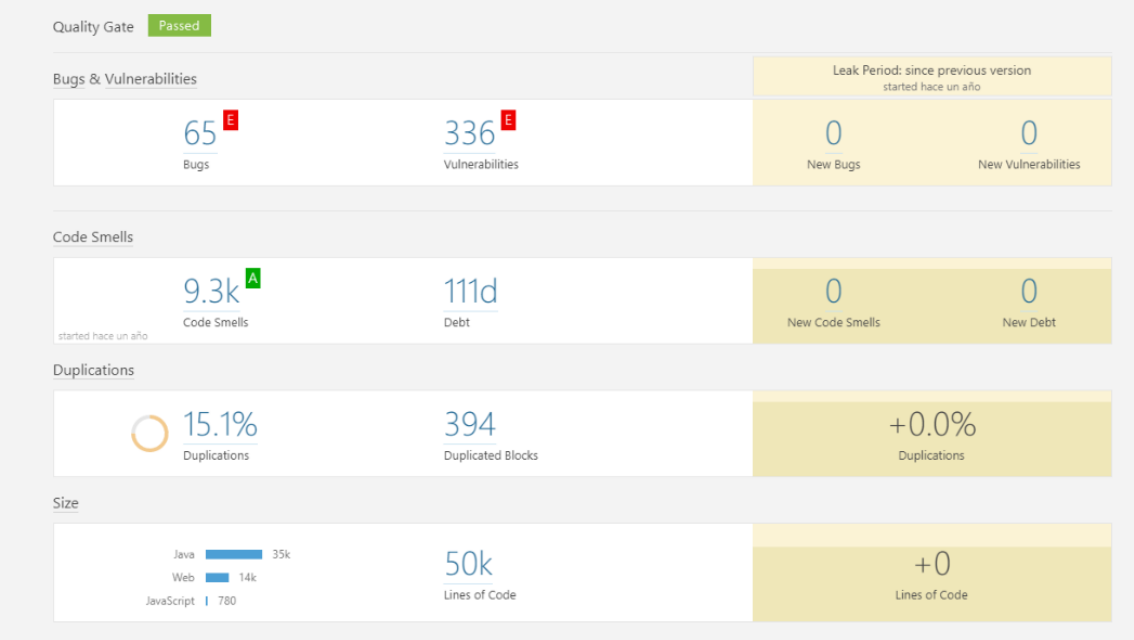


Figura Nº23 (Escaneo de proyecto sin fallas)

INDICADOR DE LOS PROYECTOS DE PRUEBA DE TIPO INCIDENCIA

LEYENDA	Julio
>= 10 Incidencias Mensual	
dencias Mensual <10 - 5> Incidencias Mer	
<= 5 Incidencias Mensual	

Mes	Cantidad
ene	13
feb	10
mar	6
abr	11
may	11
jun	6
jul	3
Total general	**60**

Frecuencia	Mensual

Total de Atención de los Proyectos de Pruebas de Tipo Incidencia

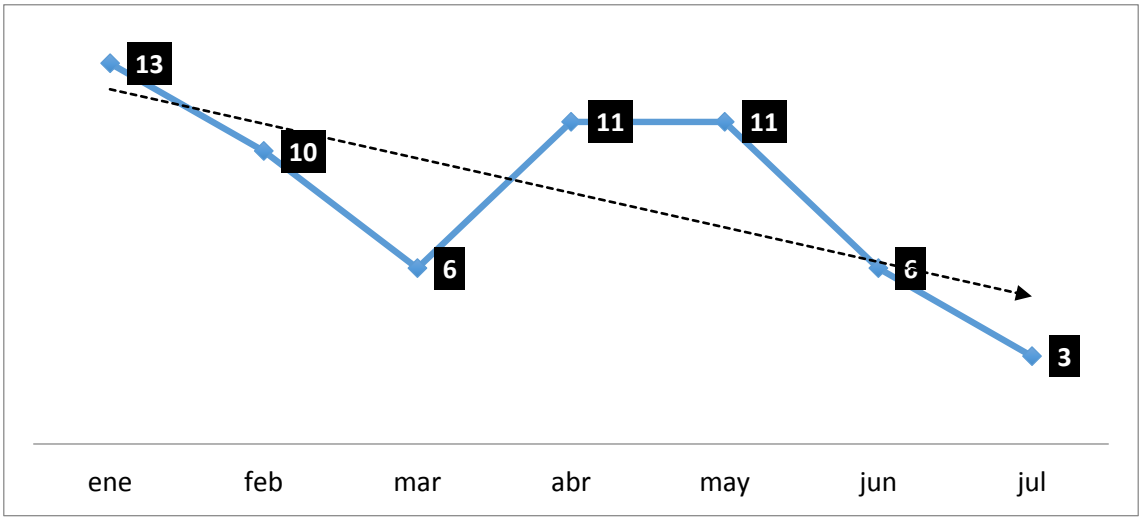


Figura Nº24 (Indicador de atención de incidencias)

- Se logró eliminar la tercerización del servicio de gestión de configuración, encargado de verificar se cumplan los estándares de calidad que muchas veces fallaba en la validación, eliminando el costo total mensual de **4,500** Soles.
- Se logró reducir el tiempo del proceso operativo de actualización de código fuente de 04 (cuatro) días a 02 (dos) horas.
- Se disminuyó la cantidad de recursos asignados para el proceso operativo de actualización de código.
- La entrega del producto o software ahora cumplen con la fecha establecida en cronograma y con la calidad que el usuario necesita.

4.1.2 Presupuesto

- **Costo**

COSTO EMPRESA 1.5

CUADRO DE ASIGNACION DE RECURSOS DE LA EMPRESA QUE EJECUTA EL PROYECTO

SUELDO	C.EMPRESA	Cantidades	Cargos	INICIO		ANALISIS Y DISEÑO		CONSTRUCCION Y PRUEBAS						IMPLANTACION		TOTALES	
				mes-1		mes-2		mes-3		mes-4		mes-5		mes-6		mes-7		
7,000	10,500	2	ANA	200%	21,000.00	100%	10,500.00	100%	10,500.00	200%	21,000.00	200%	21,000.00	100%	10,500.00	200%	21,000.00	115,500.00	
2,000	3,000	1	TES	0%	-	0%	-	0%	-	0%	-	200%	6,000.00	200%	6,000.00	0%	-	12,000.00	127,500.00
TOTALES					21,000.00		10,500.00		10,500.00		21,000.00		27,000.00		16,500.00		21,000.00	127,500.00	

NOTA: Todos los montos están expresados en nuevos soles

- **Flujo del proyecto**

	EJECUCION DEL PROYECTO							
Costos	**mes-1**	**mes-2**	**mes-3**	**mes-4**	**mes-5**	**mes-6**	**mes-7**	**total**
Personal	21,000.00	10,500.00	10,500.00	21,000.00	27,000.00	16,500.00	21,000.00	127,500.00
Otros Gastos	100	200	300	500	300	300	200	1,900.00
Total	21,100.00	10,700.00	10,800.00	21,500.00	27,300.00	16,800.00	21,200.00	129,400.00
Acumulado	21,100.00	31,800.00	42,600.00	64,100.00	91,400.00	108,200.00	129,400.00	

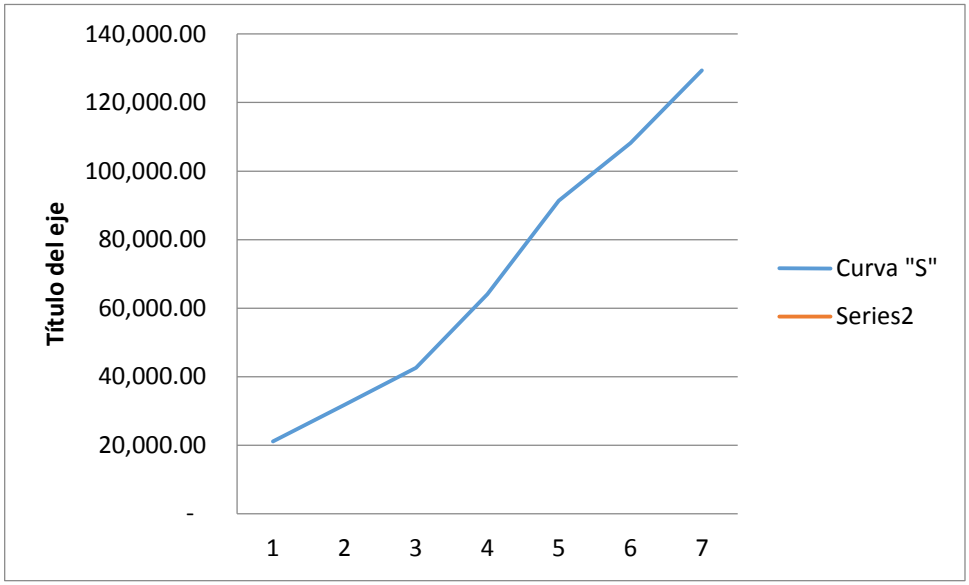


Figura Nº25 (Curva S del proyecto)

- **Evaluacion del Proyecto:**

Se realizó la evaluación económica del proyecto para conocer la rentabilidad y se obtuvo el siguiente resultado:

	Cálculo del Precio										
	Cálculo del Precio							EMPRESA QUE EJECUTA EL PROYECTO			
								FLUJO DE CAJA			
				mes-1	mes-2	mes-3	mes-4	mes-5	mes-6	mes-7	total
Costo	129,400.00		Ingresos	55,457.14		55,457.14				73,942.86	184,857.14
Margen	30%		Egresos	21,100.00	10,700.00	10,800.00	21,500.00	27,300.00	16,800.00	21,200.00	129,400.00
Precio	184,857.14		**Flujo Neto**	**34,357.14**	**-10,700.00**	**44,657.14**	**-21,500.00**	**-27,300.00**	**-16,800.00**	**52,742.86**	**55,457.14**
Utilidad	55,457.14		Acumulado	34,357.14	23,657.14	68,314.29	46,814.29	19,514.29	**2,714.29**	**55,457.14**	

Formas de Pago		
Inicio	30%	55,457.14
AyD	30%	55,457.14
Entrega	40%	73,942.86

Tasa de Dcto anual	12%	Tasa Mensual = (Tasa anual+1)^1/12 -1
Tasa de Dcto mensual	0.949%	
VANI	**S/. 181,232.65**	
VAN NETO	**S/. 53,694.62**	
Rentabilidad	**VAN NETO / VANI**	**29.63%**
	S/. 53,694.62	

El VAN del proyecto es **S/. 53,694.62 Soles** para una tasa de descuento (r) de 12% anual. Al ser este valor mayor a cero, se puede afirmar que el proyecto es rentable.

4.1.3 Cronograma

Para el presente proyecto de ISP se ha elaborado un cronograma que permite visualizar el tiempo consumido para las actividades efectuadas en el proceso de implementación de las herramientas de Integración Continua como solución al problema de gestión de código fuente, así mismo se muestran los hitos que permiten conocer si el proyecto va avanzando de acuerdo al tiempo estimado o el desfase que pueda haber ante cualquier incidencia.

Nombre de Tarea	Duración	Comienzo	Fin
PROYECTO. IC - PANDORA	**127 días**	**16/01/2017**	**07/07/2017**
Levantamiento de Información	**12 días**	**16/01/2017**	**31/01/2017**
Definición de requerimientos	3 días	16/01/2017	18/01/2017
Análisis Funcional	2 días	19/01/2017	20/01/2017
Análisis Técnico	2 días	23/01/2017	24/01/2017
Alcance del proyecto	3 días	25/01/2017	27/01/2017
Reunión con el usuario	2 días	30/01/2017	31/01/2017
Instalación y Configuración	**47 días**	**01/02/2017**	**04/04/2017**
Instalación de GIT	10 días	01/02/2017	10/02/2017
Instalación de Bitbucket	5 días	13/02/2017	17/02/2017
Instalación de SonarQube	5 días	20/02/2017	24/02/2017
Instalación de JenKins	5 días	27/02/2017	03/03/2017
Configuración de PC desarrollo	3 días	06/03/2017	08/03/2017
Configuración de PC QA	2 días	09/03/2017	10/03/2017
Definición de la plantilla de calidad de código	5 días	13/03/2017	17/03/2017
Clonación de Repositorios	5 días	20/03/2017	24/03/2017
Reunión avance con el equipo de desarrollo y QA	1 día	27/03/2017	27/03/2017
Testing QC del nuevo sistema de IC	4 días	28/03/2017	31/03/2017
Ajustes en la configuración de herramientas	2 días	03/04/2017	04/04/2017
Testing QA	**23 días**	**05/04/2017**	**05/05/2017**
Pruebas Funcionales (Ciclo I)	13 días	05/04/2017	21/04/2017
levantamiento de observaciones	5 días	24/04/2017	28/04/2017
Pruebas Funcionales (Ciclo II)	4 días	02/05/2017	05/05/2017
Comité de avance con Jefatura QA y Desarrollo	1 día	05/05/2017	05/05/2017
Documentación y Capacitación	**25 días**	**08/05/2017**	**09/06/2017**
Elaboración de Manual de Usuario QA y Desarrollo	5 días	08/05/2017	12/05/2017
Aprobación de Manuales QA y Desarrollo	3 días	15/05/2017	17/05/2017
Capacitación equipo Desarrollo	5 días	18/05/2017	24/05/2017
Capacitación equipo QA	5 días	25/05/2017	31/05/2017
Elaboración de Manual técnico de Herramientas Implem.	5 días	01/06/2017	07/06/2017
Aprobación de Manuales técnicos	2 días	08/06/2017	09/06/2017
Preparación Entorno Producción	**10 días**	**12/06/2017**	**23/06/2017**
Configuración de servidores	5 días	12/06/2017	16/06/2017
Capacitación del equipo Producción	5 días	19/06/2017	23/06/2017

Cierre	**10 días**	**26/06/2017**	**07/07/2017**
Pase a producción del Nuevo Sistema de gestión de Código	3 días	26/06/2017	28/06/2017
Reunión de entrega final de la solución	2 días	29/06/2017	30/06/2017
Documentación de Aceptación de la solución	1 día	03/07/2017	03/07/2017
Entrega de Manuales de Usuario Funcional y Técnico	1 día	04/07/2017	04/07/2017
Aprobación de Manuales y Término del Proyecto	3 días	05/07/2017	07/07/2017

- **GESTION DE LA CALIDAD:**

Con el objetivo de garantizar la calidad absoluta del producto, en el caso del presente proyecto la solución implementada, se han planteado objetivos de calidad las cuales se deben cumplir o alcanzar para certificar que el proyecto está apto para su puesta en producción.

Los objetivos de calidad planteados son los siguientes:

- ✓ El proyecto en la fase de pruebas de calidad debe tener la mínima cantidad de observaciones o en el mejor escenario cero (0) observaciones.
- ✓ De contar con observaciones, estas deben estar considerados con criticidad baja, es decir que no impacten en el correcto funcionamiento del sistema (observaciones de forma).
- ✓ La solución implementada debe estar acorde con lo definido en el alcance establecido por el usuario.
- ✓ El sistema debe cumplir lo descrito en el alcance y a su vez con el diseño técnico establecido en la fase de Análisis y Diseño, es decir alcanzar los objetivos utilizando los recursos necesarios y cumpliendo con los estándares establecidos.
- ✓ La solución implementada debe ayudar a eliminar gastos económicos innecesarios y desperdicios de un proceso influyendo de esta manera en el buen desempeño financiero de la organización.
- ✓ La reducción del tiempo de respuesta del sistema es un factor muy importante en el presente proyecto, puesto que este tiene como principal objetivo optimizar el proceso de gestión y estandarización de código fuente y por ende el proceso operativo de desarrollo de software. El proyecto ayuda a reducir la cantidad de errores encontrados en el producto detectándolos de forma temprana y corrigiéndolos rápidamente, antes de implementar la solución el proceso de actualización de código fuente en el área QA tardaba 3 días luego de la implementación se redujo el tiempo a 20 minutos.

Considerar que los objetivos de calidad deben cumplir con las siguientes características principales:

- Que sean claros.
- Que sean medibles.
- Que sean alcanzables.
- Que sean motivadores.

Si el sistema de Integración Continua implementado no cumple con los objetivos de calidad planteados, se considera un producto no apto para ser puesto en producción, en algunas ocasiones se vuelve a evaluar la viabilidad del proyecto o se cancela de manera definitiva.

En el presente proyecto se cumplieron cada uno de los objetivos de calidad planteados y se concluyó satisfactoriamente con el pase a producción.

- **Elaboración del EDT:**

Se ha elaborado el siguiente EDT para mayor visiblidad de las actividades a realizarse a lo largo del proyecto:

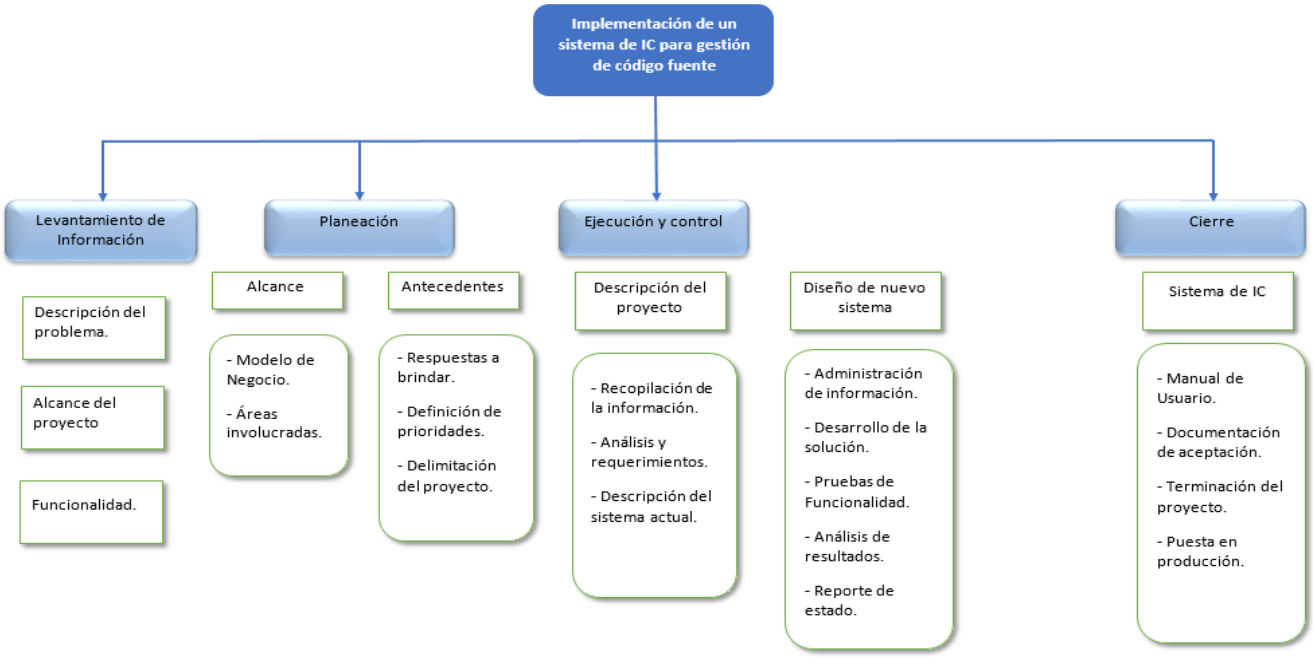


Figura Nº26 (EDT)

ORGANIGRAMA:

El siguiente organigrama muestra la jerarquía de los recursos involucrados en el desarrollo del presente proyecto:

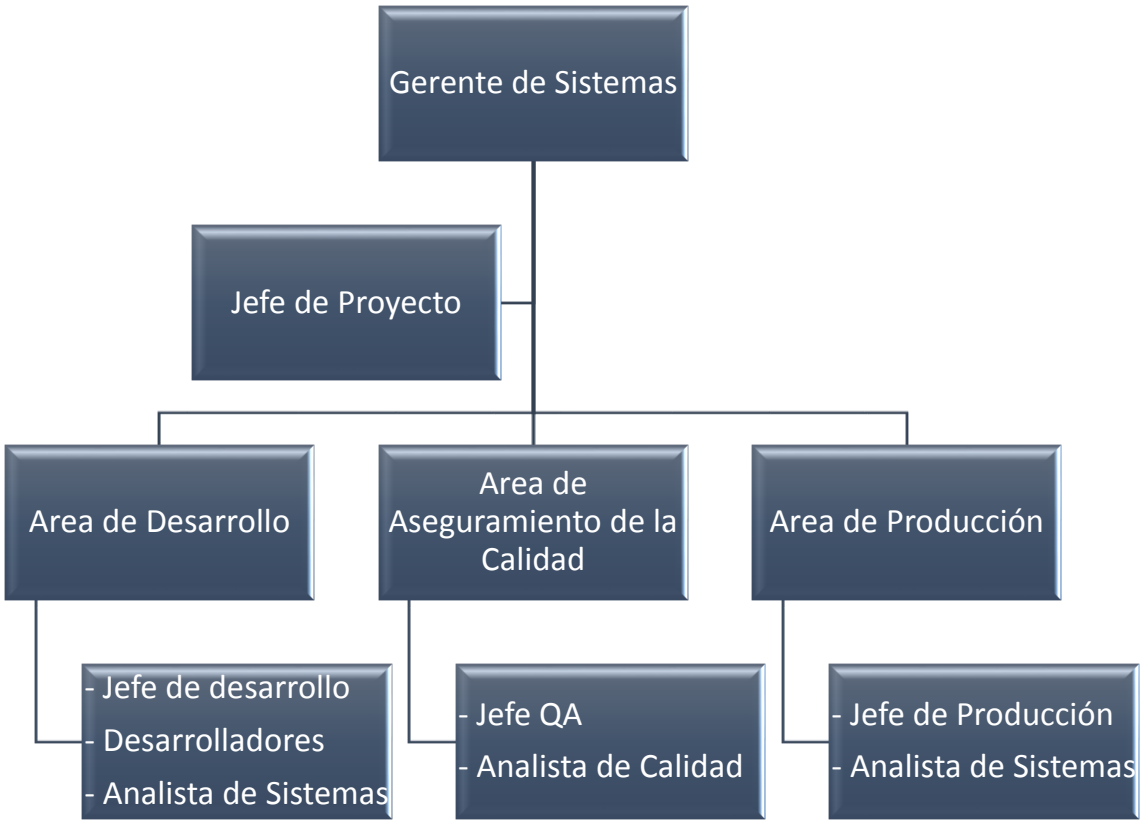


Figura Nº27 (Organigrama)

Gerente de Sistemas:

Su responsabilidad está relacionada al correcto funcionamiento del sistema, las adquisiciones de equipos informáticos y a la toma de decisiones estratégicas.

Jefe de Proyecto:

Responsable absoluto del planeamiento y ejecución del proyecto, tiene la capacidad de reconocer los riesgos que afectan al éxito del proyecto, debe realizar constantes mediciones de factores internos o externos que sean considerados fuentes de riesgo. Es un agente de cambio, capaz de reconducir al equipo para convertir las debilidades en fortalezas y así culminar el proyecto satisfactoriamente.

Encargado de enviar el avance del proyecto cada cierto periodo a los interesados del proyecto.

Jefe de Desarrollo:

Encargado de supervisar diariamente al personal involucrado en el proyecto y las actividades del departamento bajo su responsabilidad, organiza el proyecto a desarrollar tomando en cuenta las necesidades de informáticas de los usuarios y recursos existentes.

Encargado de evaluar la factibilidad del proyecto, el alcance y objetivos, además de asignar tareas y recursos para su desarrollo.

Desarrolladores:

Responsables de verificar que la solución implementada en el área de desarrollo para optimizar el proceso de gestión de código fuente y por consecuente el proceso de desarrollo de software permita poder realizar con facilidad y sin restricciones sus actividades diarias.

Analista de Sistemas:

Es el responsable de verificar que la implementación de la solución en el área de desarrollo sea el óptimo para la ejecución de las actividades de los desarrolladores, así como supervisar que las herramientas y procedimientos ejecutados no afecten los equipos informáticos que tienen a su disposición, realiza constantes revisiones técnicas del proyecto.

Jefe QA:

Responsable directo de la certificación de calidad del proyecto, con el poder de determinar la continuidad de desarrollo del proyecto. Encargado de hacer cumplir las políticas de calidad y los objetivos propuestos.

Analista de Calidad:

Es el responsable de certificar la calidad absoluta de la solución a implementar, realiza las pruebas del sistema de Integración Continua en su totalidad es decir valida el comportamiento del sistema desde el desarrollo del software hasta el pase a producción del producto. Encargado de informar las no conformidades al responsable del desarrollo del proyecto para su revisión y corrección. Realiza las pruebas del sistema basándose en estándares y objetivos de calidad así como el alcance realizado por el usuario.

Jefe de Producción:

Responsable de hacer el seguimiento de las actividades realizadas por el analista encargado de ejecutar el pase a producción del producto, responsable directo del correcto funcionamiento del sistema de integración continua implantado en el entorno de producción.

Analista de Sistemas de Producción:

Responsable de ejecutar el pase a producción del sistema de IC en el entorno productivo, así como validar el correcto desempeño del sistema implantado e informar a las áreas de desarrollo y calidad la detección de errores encontrados en las actividades de pase a producción de distintos softwares.

- **GESTION DE LAS COMUNICACIONES:**

Los responsables de transferir la información de avances del proyecto a las diferentes partes interesadas son los jefes de cada área de sistemas: Jefe de desarrollo, Jefe de Calidad (QA), Jefe de Producción y Jefe de Proyecto, toda la información recolectada y almacenada de cada área es derivada y centralizada al Gerente de Sistemas para la toma de decisiones correspondiente.

Los medios por los cuales se comunica el estado o el avance del proyecto son tanto formales como informales, es decir mediante reuniones y comités donde participan los actores mencionados previamente y que están directamente involucrados en el proyecto. Los acuerdos definidos en cada comité son plasmados en Informes digitales los cuales son enviados vía correo electrónico a todos los involucrados en el proyecto.

- **STAKEHOLDERS:**

En el desarrollo del siguiente proceso se han identificado a las personas que participan directa e indirectamente en cada etapa del proyecto y que podrían ser afectadas positiva o negativamente por el potencial impacto en el éxito del proyecto.

- **Gerente de Sistemas:** Es el sponsor del proyecto, quien proporciona los recursos y apoyo para el proyecto y que es el responsable de facilitar su éxito.
- **Jefe de Proyecto:** Interesado en que el proyecto a su cargo se culmine satisfactoriamente en el tiempo establecido.
- **Usuarios:** Son los beneficiados directos por el proyecto, son las personas que van a usar el sistema resultante de Integración Continua y que por lo tanto van a mostrar su satisfacción o insatisfacción con este.
- **La organización que ejecuta el proyecto:** Son los beneficiados económicamente por el servicio brindado al cliente, aquí no solo están los integrantes del desarrollo del proyecto sino también el gerente general, el supervisor y la dirección interna.

- **ADQUISICIONES:**

El presente proyecto no cuenta con un informe de adquisiciones dado que se basa en un servicio otorgado al cliente (organización que recibe la solución) quienes son los responsables de otorgar a la organización que ejecuta el proyecto los recursos necesarios para el desarrollo del mismo.

Conclusiones:

- Existen, hoy en dia, numerosas soluciones que se adaptan a distintas necesidades y que presentan diferentes costes y requerimientos. Es por este motivo que es importante analizar bien la oferta y determinar cual es la herramienta más adecuada a cada caso.
- En la empresa se ha optado por seleccionar las herramientas: Git, SonarQube, Jenkins y Bitbucket por ser herramientas que se adecuan a las necesidades y posibilidades de la organización, además por las distintas características amigables de las herramientas y por la funsionalidad de cada uno.
- La implantación ha sido realizada luego de la fase de pruebas en la que se ha terminado de configurar, ajustar y comprobar todo lo necesario de manera que se determinó la adecuación completa a las necesidades del cliente.
- Luego de instaladas las herramientas, quedó listo para su uso y accesible desde cualquier equipo de los desarrolladores y analistas de la organización.
- Se puede afirmar que se ha tenido un cierto impacto sobre los colaboradores o usuarios de la solución ya que adaptarse a una nueva herramienta no es fácil ni rápido, por ello se realizaron las capacitaciones obteniendo buenos resultados.
- Por último, se han enviado convocatorias de reuniones a las áreas del departamento de sistemas para presentarles las herramientas y el uso que se le pretende dar, posterior a estas reuniones se han realizado algunos ajustes finales para una mayor adaptación y corrección de errores.
- Luego de algunos meses de utilización de las herramientas y del nuevo flujo de actualización de código fuente propuesto se van viendo las mejoras en el proceso de gestión de código y por ende en la calidad del producto final.

Anexos:

Estimados,

Se remite el cronograma con las actividades a realizarse en el proyecto de DevOps.

Así mismo, se brinda respuesta a las consultas del Analista 01:

Pandora proporcionará los requerimientos técnicos (s.o, cpu´s, memoria entre otros) recomendados para la instalación/configuración del producto Artifactory.

➔ Es conforme, se entregará esta especificación

Confirmar si los despliegues automatizados para WARI y Factrack consideran la creación de colas locales y remotas.

➔ Para esta primera fase del proyecto, como se indicó en la presentación, solo se considera despliegue de aplicaciones web en el WAS, para que se pueda ver el beneficio de un proyecto DevOps de manera rápida. En una segunda fase se podría considerar esos tipos de servidores.

Manuales de instalación y configuración de todos los componentes de la solución.

➔ Se está considerando en el cronograma **"Elaboración de Manual de Configuración de Jenkins"** donde se explicará cómo configurar desde cero el Jenkins y cómo se construirá los pipelines

En el caso de desarrollo, necesitaríamos manual operativo indicando configuraciones realizadas en los servidores WAS (en caso existiera) y la secuencia de pasos de lo que se debería revisar en caso se presenten problemas. Considerar lista de puertos.

➔ El manual operativo que se indica aplicaría si se modifica el aplicativo. Si en la fase preliminar de revisión de los aplicativos se identifica que no se puede compilar y desplegar por comandos algún aplicativo, se debería de atender de manera externa dichas modificaciones, es decir, en el cronograma adjunto no se encuentra esas actividades.

Quedamos atentos, a cualquier otra consulta o indicación de inicio del proyecto.

Saludos,

Anexo 1 (Correo de coordinación con el cliente)

Conforme con la propuesta sólo quisiera hacer una precisión para clarificar los entregables que vamos a recibir, tal vez podamos actualizarlos en la sección de Alcance o en una nueva página.
Favor tus cometarios en cada punto para coordinar el inicio del proyecto la siguiente semana.
Gracias.

- Definición de estándares para:
  - Versionamientode Proyectos
  - Librerías Compartidas
  - Jobs de Ejecución

QA: Confirmar la entrega de un documento por cada uno de los estándares

- Configuración de Servidores:
  - Servidor de Versiones de Fuentes (GIT)
  - Servidor de Librerías Compartidas (Artifactory)
  - Servidor de DevOps(Jenkins)

QA: Confirmar la entrega de un documento de configuración por cada uno de los servidores

- Creación de Pipelines para los aplicativos
  - Wari(DESA,QC,QAInternoyQAExterno)
  - Factrack(DESA,QC,QAInternoyQAExterno)
  - Pagares(DESA,QC,QAInternoyQAExterno)

QA: Confirmar la entrega de un documento de monitoreo de cada pipeline o un documento de configuración de la implementación

- Se dejará configurado la integración con SONAR y JUnit

QA: Confirmar la entrega de un documento de configuración de Sonar y JUnit

- Se proporcionará las especificaciones técnicas (SO, CPU´s, memoria, entre otros) recomendados para la instalación/configuración del producto Artifactory.

QA: Confirmar la entrega del documento de especificaciones técnicas de cada parte de la arquitectura y la relación de configuraciones y puertos entre estos componentes.

- Se considera despliegue de aplicaciones web en el WAS. En una siguiente fase se podría trabajar con BUS y MQ.

QA: Confirmar la entrega de un manual, checklist o especificaciones del despliegue de cada aplicación.

Anexo 2 (Correo de coordinación con el cliente)

A continuación se responde a tus consultas.

En dichas respuestas se indica los puntos que sí están considerados en la propuesta enviada, así como también aquellos que no están. Confirmar si se requerirán, para proceder a actualizar el cronograma.

Confirmar si con las respuestas se absuelven todas las dudas o consultas, para actualizar la ppt de la propuesta considerando el detalle que estamos respondiendo.

Quedo atenta a cualquier comentario.

- Definición de estándares para:
  - Versionamientode Proyectos
  - Librerías Compartidas
  - Jobs de Ejecución

QA: Confirmar la entrega de un documento por cada uno de los estándares

PANDORA: Dentro de la propuesta ya se está considerando los siguientes documentos de estándares

- Elaborar "Documento de Estructura del Repositorio"
- Elaborar "Documento de Guía de Proceso de Versionamiento"
- Elaborar "Documento de Estándares de Repositorio y Nomenclatura de Componentes"
- Elaborar "Documento de Estándares de Jenkins"

QA: CONFORME

- Configuración de Servidores:
  - Servidor de Versiones de Fuentes (GIT)
  - Servidor de Librerías Compartidas (Artifactory)
  - Servidor de DevOps (Jenkins)

QA: Confirmar la entrega de un documento de configuración por cada uno de los servidores

PANDORA: Dentro de la propuesta se entrega el documento de:

- Elaboración de Manual de Configuración de Jenkins

Los otros documentos solicitados **no están dentro de la propuesta**, pero se agregar la actividad de "Elaboración de Librerías Compartidas (Artifactory)", el cual sería 18 horas adicionales.

El de GIT no se considera porque dicha herramienta ya lo tienen configurado e instalado.

QA: CONFORME, ACTUALIZAR EN LA PROPUESTA Y CRONOGRAMA, EN LO QUE CORRESPONDE A GIT SÒLO APLICARIA EN CASO SE HAYA HECHO ALGUNA CONFIGURACIÒN.

- Creación de Pipelines para los aplicativos
  - Wari (DESA,QC,QAInternoyQAExterno)
  - Factrack (DESA,QC,QAInternoyQAExterno)
  - Pagares (DESA,QC,QAInternoyQAExterno)

- Se identificará las diferencias de arquitectura tecnológica de los aplicativos WARI, Factrack y Pagares, en los ambientes DESA, QC, QA Interno y QA Externo.

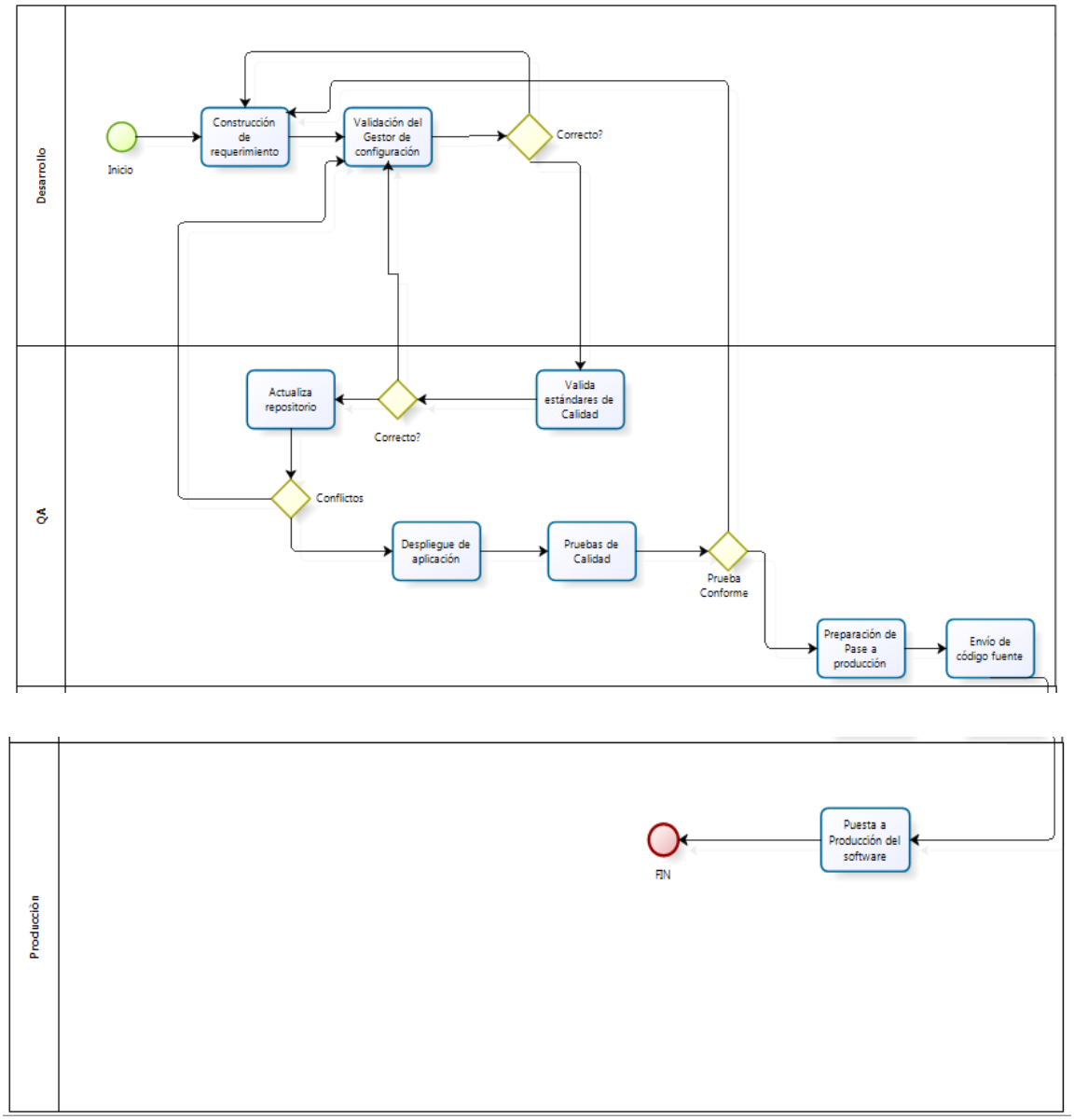
QA: Los documentos solicitados en los puntos previos deben considerar estas diferencias y precisiones por cada ambiente.

PANDORA: No **se había considerado este punto en la propuesta**. De requerirlo serían 23 horas adicionales, lo cual involucraría las actividades de "Elaborar Documento de Diferencias de Entornos" y "Elaborar Presentación de diferencias de Entornos"

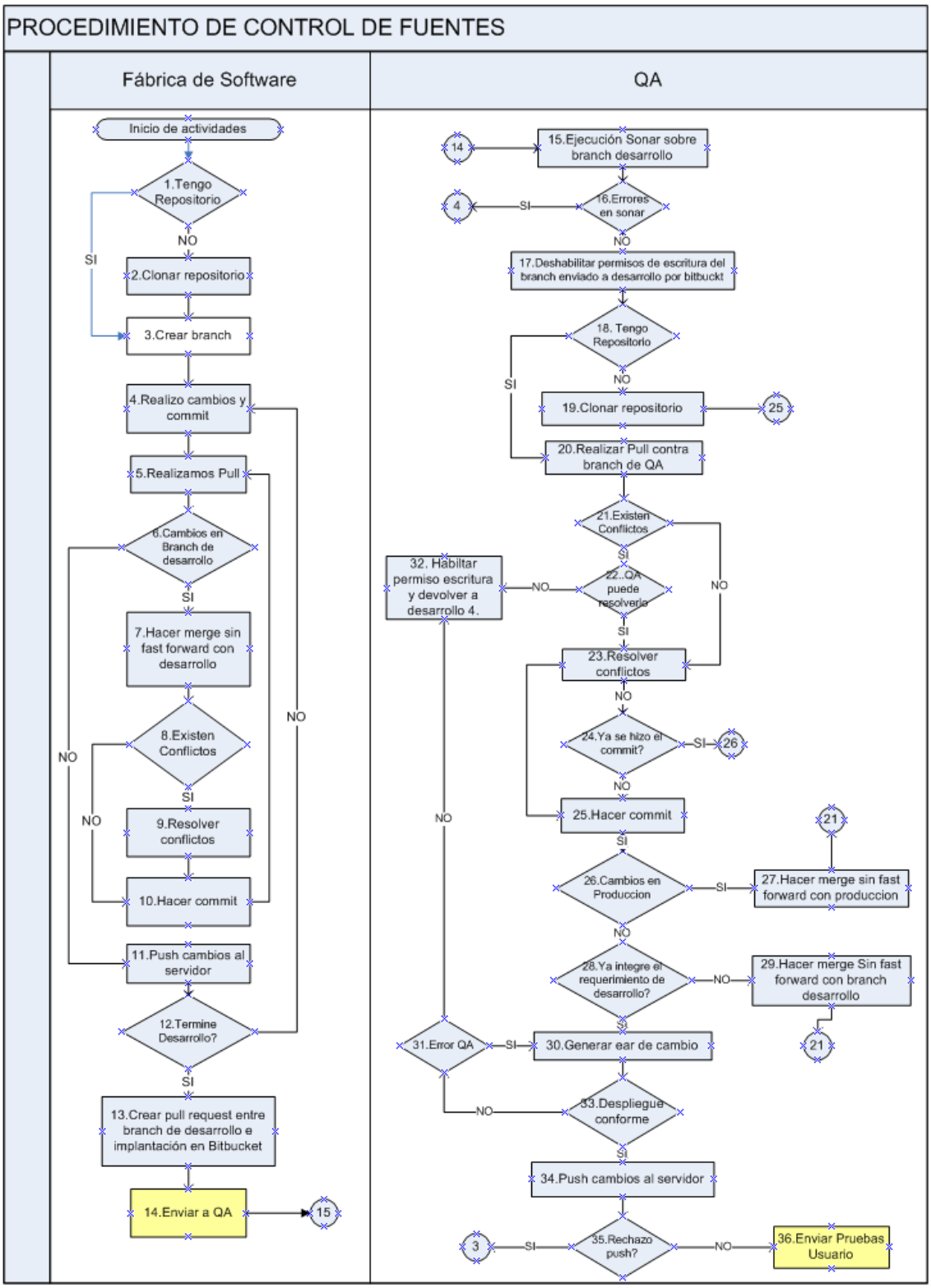
QA: CONFORME ACTUALIZAR EN LA PROPUESTA Y CRONOGRAMA

Saludos,

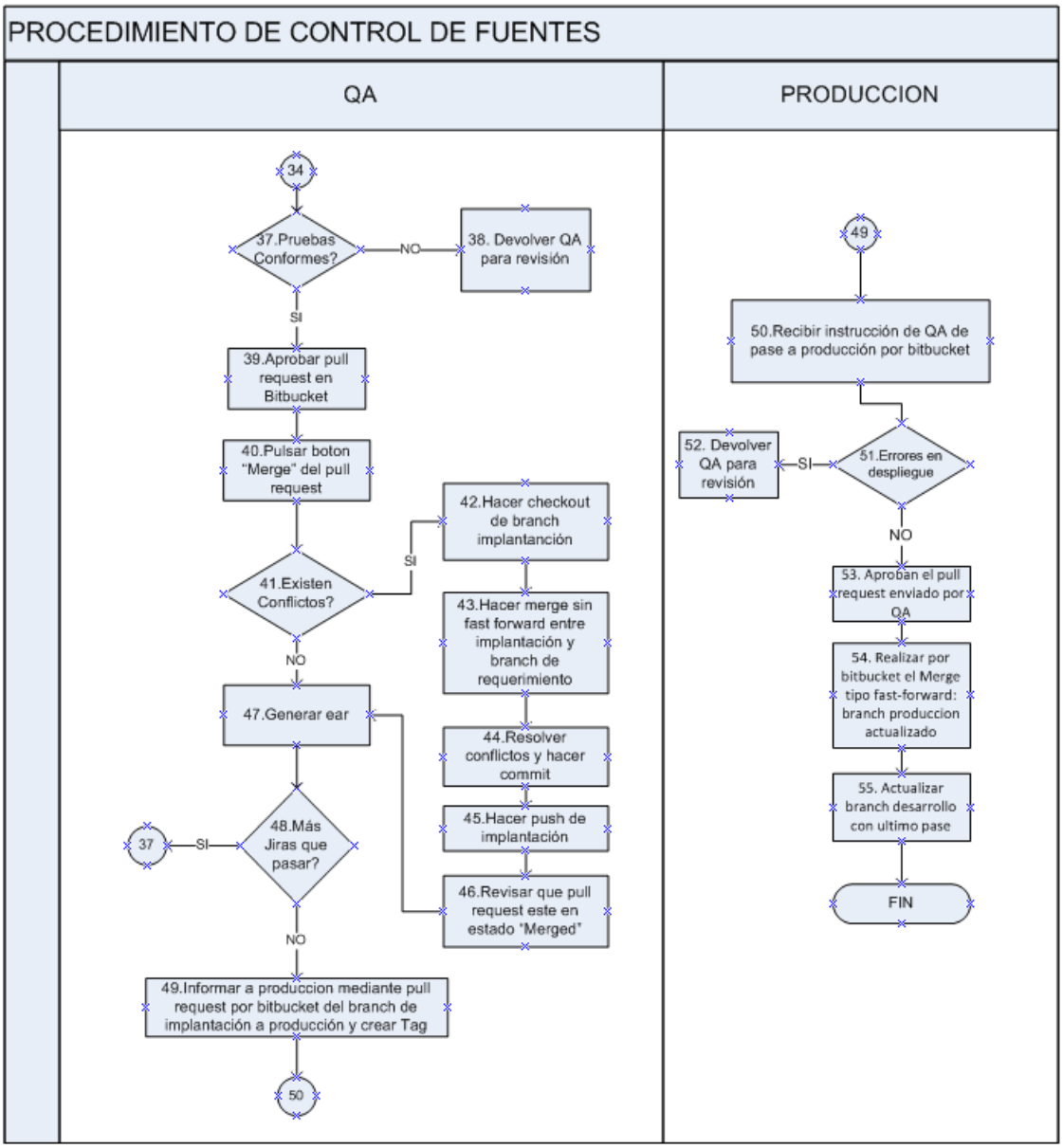
Anexo 3 (Correo de coordinación con el cliente)



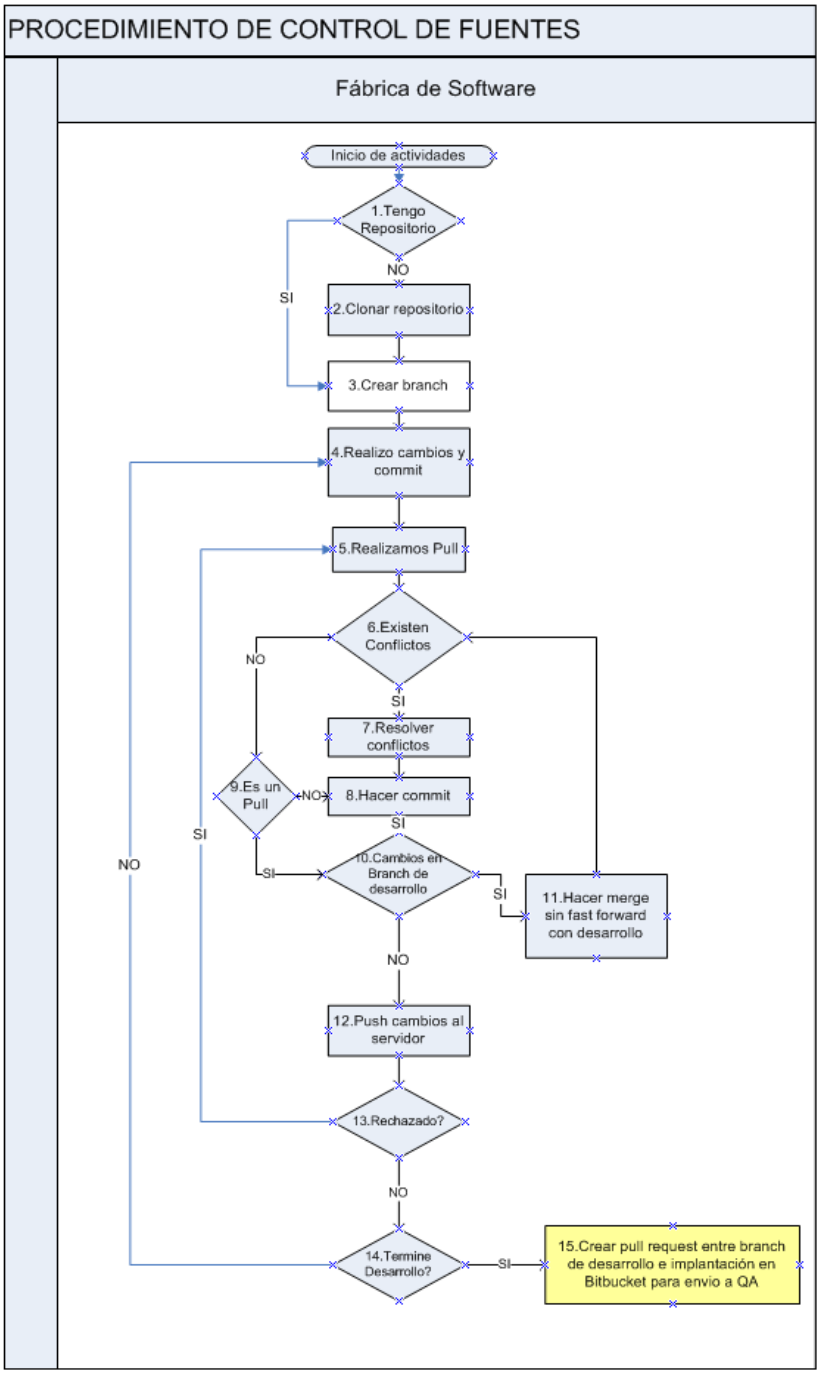
Anexo 4: (Procedimiento de actualización de código antes de la solución)



Anexo 5 (Escenario I – actualización de código)



Anexo 6 (Escenario I – actualización de código)



Anexo 7 (Escenario II – Control de fuentes para un requerimiento o proyecto con más de un desarrollador.)

Bibliografía:

Blé Jurado, Carlos (2010). *Diseó Ágil con TDD*, Edit: lulu.com

Bangalore, Pathania Nikhil (2017). *Pro Continuous Delivery with Jenkins 2.0.* Karnataka India: Apress.

Hinostroza, Berta (2014). Incorporación de la integración continua en el desarrollo de software. Recuperado de https://pirhua.udep.edu.pe/bitstream/handle/11042/1826/MAS_DET_006.pdf?sequence=1

Huércano Ruiz, F. y Villar Cueli, J. (2014). Implementación *e integración de elementos software con tecnologías basadas en componentes IFCT0609*. IC Editorial

López Carlos. (2001, noviembre 11). *Aseguramiento de la calidad y sistemas de calidad*. Recuperado de https://www.gestiopolis.com/aseguramiento-calidad-sistemas-calidad/

Paoletta, Martin (2016) El Cronista. Una *Techie que se anticipó al mercado. RedBee*. Recuperado de https://www.cronista.com/pyme/Una-techie-que-se-anticipo-al-mercado-20161222-0012.html

Herrera Perez, S. (2017). *Proyecto Fin de carrera Análisis e Implementación de la Integración Continua*. Recuperado de https://repositorio.unican.es/xmlui/bitstream/handle/10902/12058/396597.pdf?sequence=1